SonitusBE

Acoustic Boundary Element Solver.

Dr Dan J. O'Boy

http://www.djoboy.co.uk

R1

All rights reserved.

This manual is provided for reference and may not be copied and / or altered without the permission of the author. References to the manual should be made whenever material is used.

The manual provides guidance and information in order to use the Sonitus BE program and may be altered without notice being provided. No responsibility is assumed or given for any damage, commitments, liabilities or inaccuracies incurred as a result of the information provided.

All material is the intellectual property of this company through the owner Dr Dan J. O'Boy, who applies the property rights. The terms and conditions are applied through and in accordance with English law and any disputes will be subject to the jurisdiction of the courts of England and Wales of the United Kingdom.

Revision 1

Copyright ©2008

Dr Dan J O'Boy
4 The Green
Exton
Oakham
Rutland
LE158AP

United Kingdom
enquires@djoboy.co.uk

SonitusBE

TABLE OF CONTENTS

TABLE OF CONTENTS 5

LIST OF FIGURES 7

LIST OF TABLES 9

I. INTRODUCTION 11

II. REQUIREMENTS 12

III. BACKGROUND OF THE BOUNDARY ELEMENT METHOD 13

IV. NOMENCLATURE 14

V. KEY TERMINOLOGY 15

VI. MATHEMATICAL FOUNDATION OF THE BOUNDARY ELEMENT METHOD 20

PRINCIPLE OF THE INDIRECT BOUNDARY ELEMENT METHOD 23
DISCRETISATION OF THE SURFACE DOMAIN 25
INTEGRATING A BOUNDARY ELEMENT 27
QUADRILATERAL ELEMENTS 27
TRIANGULAR ELEMENTS 29
SOURCE DEFINITION AND LOCATION 30
DATA RECOVERY PARAMETERS 30
THE SOLUTION METHOD FOR A MATRIX PROBLEM 34

VII. INPUT FILE DEFINITION 34

VIII. OUTPUT FILE DEFINITION 36

IX. PROGRAM OVERVIEW 37

SONITUSBE.CPP 37

X. SUBROUTINE FUNCTIONS 38

ERRORHANDLE.H (COMPLETE) 38
INTERP.H (COMPLETE) 38
MESSAGES.H (COMPLETE) 40
SOLUTIONCHOICE.H (COMPLETE) 40
VERIFICATION.H (COMPLETE) 41
STRUCTURES.H (COMPLETE) 41
FILEREADERSONITUS.H (COMPLETE) 42

RESULTOUTPUT.H (COMPLETE) 43
COMPLEX.H (COMPLETE) 44
INDIRECTSOLVER.H (COMPLETE) 44
MAKEFILE (COMPLETE) 46
DJO_LU.H (INCOMPLETE) 46
GAUSSJC.H (COMPLETE) 47

XI. SAMPLE FILES 47

SOLID GEOMETRY 48
INFINITE PLANE SURFACE 48
DATA RECOVERY MESH (EXTERIOR) 49
DATA RECOVERY MESH (INTERIOR) 49
FREE-FIELD SOLUTION 50
SOUND PRESSURE IN A DOMAIN WITH A SOLID GEOMETRY 52
SOUND PRESSURE INSIDE AN ENCLOSURE WITH DAMPING MATERIAL INCLUDED ON ONE SURFACE 52

XII. NOTES 52

EXTERIOR AND INTERIOR DOMAINS 52
ELEMENT TYPES 53
SOUND SOURCES 53
INFINITE GROUND PLANE 53

XIII. INPUT FILE TYPES - DAT 53

XIV. DEBUG REQUIREMENTS 53

XV. SUMMARY 53

XVI. REFERENCES 55

LIST OF FIGURES

FIGURE 1 A NODE CHARACTERISED BY THREE INDEPENDENT COORDINATES 16
FIGURE 2 A TRIANGULAR ELEMENT CHARACTERISED BY THREE INDEPENDENT NODES, AND A QUADRILATERAL ELEMENT CHARACTERIZED BY FOUR. NOTE THAT THE DIRECTION NORMAL OF ALL ELEMENTS MUST BE CONSISTENT. 16
FIGURE 3 OMNI-DIRECTIONAL SOUND PRESSURE PROPAGATING FROM A MONOPOLE FUNDAMENTAL SOLUTION. 19
FIGURE 4 DIRECTIONAL SOUND FIELD PROPAGATING FROM A DISTRIBUTION OF MONOPOLES AT LOW AND HIGH FREQUENCIES. 20
FIGURE 5 DEFINITION OF A DOMAIN FOR ACOUSTIC CALCULATIONS. 21
FIGURE 6 DISCRETISATION OF THE SURFACE BOUNDING THE SOLUTION DOMAIN. THE SURFACE IS CREATED BY JOINING NODES TOGETHER WITH LINEAR SEGMENTS. WHEN COMPLETED OVER A THREE DIMENSIONAL DOMAIN, THE LINEAR SEGMENTS CREATE SURFACE AREAS (ELEMENTS). 23
FIGURE 7 BOUNDARY INTEGRAL SURFACE ILLUSTRATING HOW EACH ELEMENT TAKES INTO ACCOUNT THE EFFECT OF A SOURCE OF UNKNOWN AMPLITUDE LOCATED AT EACH ELEMENT. THE PROBLEM IS THEN POSED AS "WHAT AMPLITUDE OF ALL OF THE SOURCES YIELDS THE CORRECT KNOWN BOUNDARY CONDITION AT THIS ELEMENT?" IN MATRIX FORM. 26
FIGURE 8 QUADRILATERAL BOUNDARY INTEGRAL SURFACE SHOWING NON-DIMENSIONAL COORDINATE FORM OF THE SURFACE DIRECTION. 28
FIGURE 9 QUADRILATERAL LINEAR INTEGRATION METHOD. 28
FIGURE 10 TRIANGULAR LINEAR INTEGRATION METHOD. 29
FIGURE 11 BOUNDARY INTEGRALS FROM THE SURFACE TO THE DATA RECOVERY MESH. 32
FIGURE 12 GENERATION OF A SURFACE MESH USING COMPUTER AIDED DESIGN TOOLS 35
FIGURE 13 EXAMPLES OF SOLID GEOMETRY DEFINED BY AN UNSTRUCTURED SURFACE MESH COMPOSED OF TRIANGULAR ELEMENTS AND ONE WITH A MIXTURE OF QUADRILATERAL STRUCTURED ELEMENTS (ON THE WHEEL ARCH) AND A MIXTURE OF TRIANGULAR AND QUADRILATERAL ELEMENTS ON THE WHEEL SURFACE. 48
FIGURE 14 GROUND PLANE (SIMILAR TO AN IMAGE SOURCE) 49
FIGURE 15 DATA RECOVERY MESH FOR AUTOMOBILE TYRE NOISE 49
FIGURE 16 SOUND PRESSURE FOR A SOURCE IN THE FREE-FIELD (NOTE THE SCALE OF THE AMPLITUDE IS DIFFERENT AT EACH FREQUENCY). 51
FIGURE 17 DIVERGING SURFACES LEADING TO A HORN GEOMETRY WHERE SOUND SOURCES MAY BE AMPLIFIED 52
FIGURE 18 SOUND PRESSURE FOR A SOURCE WITH A SOLID GEOMETRY IN THE DOMAIN (NOTE THE SCALE OF THE AMPLITUDE IS DIFFERENT AT EACH FREQUENCY). 52

LIST OF TABLES

TABLE 1 MINIMUM RECOMMENDED COMPUTER SPECIFICATIONS....12
TABLE 2 COMPILER RECOMMENDATIONS FOR SOURCE CODE MODIFICATION....13
TABLE 3 NOTATION FOR GREEK NOTATION....14
TABLE 4 NOTATION FOR ENGLISH NOTATION....15
TABLE 5 FUNDAMENTAL SOLUTIONS FOR ACOUSTIC WAVE PROPAGATION IN FREE SPACE....18
TABLE 6 INPUT FILE DESCRIPTION....36
TABLE 7 OUTPUT FILE DESCRIPTION....37

I. INTRODUCTION

Sonitus BE is a versatile boundary element solver for acoustic engineering problems. It can provide the sound pressure, fluid velocity or intensity in an enclosure or in the free field for a range of problems.

Boundary Element Methods (BEM) are a class of numerical solution method which are efficient for the solution of acoustic problems and differ when compared to finite element methods as they do not require a volume mesh to be generated, only a surface mesh, reducing the complexity of the problem domain.

This document details some of the technical code comprising the Sonitus BE program, including subroutine calls. The program is available in Windows or Linux specific versions. Sonitus BE is the numerical solver that takes the input data files describing the acoustic problem and constructs the matrix problem. It then automatically solves this matrix problem and uses the data recovery points specified in the input file to generate a results file.

The program runs as a standalone application, without graphics interface. This deliberately allows the user to choose the graphical user interface and results processing software from a range of sources.

- Commercial CAD packages.
- Commercial meshing packages.
- Matlab standalone code for input geometry. (Recommended)
- Sonitus Pre-BE input file processor. (Recommended)
- Commercial result processing software.
- Matlab standalone code for results processing. (Recommended)
- Sonitus Post-BE result processing software. (Recommended)

In this manual, the requirements of the boundary element program are provided in terms of the minimum recommended specifications in section II. A background of the different methods available for the study of acoustic problems is provided in section III, which shows the advantages (and disadvantages) of the boundary element method.

As the boundary element method is described in this manual is mathematical terms, a nomenclature is provided in section IV and a terminology page summarising important concepts in section V.

The mathematical formulation of a boundary integral method is then described in section VI, specifically the indirect boundary element method, the method of discretising the surface of the geometry in the acoustic domain, the mathematical routines used to integrate a surface element (both quadrilateral and triangular). The source location and amplitude are then described as are the data recovery parameters. A numerical routine for the solution to the matrix problem is briefly summarised.

As the Sonitus BE program is a stand-alone console based program, the input and output file definitions are described in sections VII and VIII respectively.

An overview of the program code is provided, so that a user may understand the organisational hierarchy of the program (section IX), functions and subroutines (section X).

A number of sample files are provided for reference and are detailed in section XI. Notes on the program are found in section XII. A summary of this manual is provided in section XV.

II. REQUIREMENTS

The requirements are highly dependent on the geometry size, mesh density and frequency range which is required. The program runs as a standalone application, without graphics interface and can therefore run either as a background activity or offline, monitored through a network connection on a server. The recommended starting points are shown in Table 1.

For pre-processing requirements, Table 1 provides useful guidance, however for other commercial packages, their minimum specifications should be consulted.

For post-processing in Matlab in Linux, it is recommended that a good graphics card be included, with OpenGL support.

ATTRIBUTE	MINIMUM SPECIFICATION
MICROSOFT WINDOWS	WINDOWS XP, SERVICE PACK 3
LINUX	OPENSUSE 10.0
MEMORY (DEPENDENT ON GEOMETRY SIZE AND MESH DENSITY)	2GB RAM MINIMUM
DISK STORAGE SIZE FOR OUTPUT FILES	200MB
PROCESSOR	INTEL CORE 2 DUO CPU2GHZ
GRAPHICS (FOR POSTPROCESSING)	MATLAB V7

Table 1 Minimum recommended computer specifications

To compile the source code in Linux, type "*make*" at the console.
To compile the source code in Windows using MinGW type "g++ SonitusBE.cpp -o SonitusBE.exe"
Alternatively, type "*make*" at the console.
To compile using operating specific directives (used for pausing the screen to show an error message), include the flag –dWINDOWS or –dLINUX in the compile command string. See makefile for example.

Under Windows XP and using the MinGW gcc compiler, it is possible to create a structure which is large enough to cause memory issues. The solution is to increase the available stack size by supplying the linker with a sufficiently large maximum memory allocation so that a stack overflow never occurs. This is currently available for MinGW by including the flag -Wl,--stack=1073741824 at compile time. This option leads to a maximum Windows stack size of 1Gb.

Note that due to dynamic allocation, Linux does not require this maximum stack size to be increased. The computer code has been written using dynamically allocated memory rather than static matrix sizes, and through passing a memory reference (pointer) to subroutines rather than copying variables, however, engineering problems typically require the solution of large matrices. While there are methods to decompose large matrices into smaller sizes for more appropriate handling, this requires application specific analysis which can often lead to longer development time for diminishing returns.

For compiler recommendations using modified source code, see Table 2.

OPERATING SYSTEM	COMPILER
MICROSOFT WINDOWS	MINGW PORTING GCC
LINUX	GCC (C++ IMPLEMENTATION)

Table 2 Compiler recommendations for source code modification

The command line is SonitusBE INPUTFILE OUTPUTFILE

For Microsoft Windows, running in a console, the typical output would be
SonitusBE.exe SampleFiles\FreeField.dat output.rslt or
SonitusBE.out SampleFiles\FreeField.dat output.rslt

The output is directed to the screen by default. To direct this to an output file,
SonitusBE.exe SampleFiles\FreeField.dat output.rslt > screenoutput.txt

III. BACKGROUND OF THE BOUNDARY ELEMENT METHOD

There are several different methods available for the prediction of the acoustic field in either an enclosed interior space or an exterior domain. These methods are usually designed to support experimental measurements or hypothetical predictions. The main prediction tools for the prediction of the acoustic field include theoretical models, finite elements methods (FE), ray methods / geometrical acoustics approximations, statistical energy analysis, variational energy methods or the boundary element method. The main features leading to advantages when the methods are compared are provided in this section.

Finite element (FE) methods are the most widely used, due to the efficient and simple integration of computer aided design tools in the engineering workplace. As computer aided engineering design usually involves the creation of CAD drawings of the object, it is easy to take those drawings and incorporate them into the acoustic analysis. Finite element tools require the discretisation of a full volume and subsequent meshing of that volume. Although it produces extremely good solutions, it can tend to be complex, requiring high computing power and for higher frequency prediction, accurate geometry. The computing requirement increases with the upper analysis frequency and size of the geometry.

The statistical energy method was designed to produce approximate acoustic prediction for segmented rooms using diffuse field assumption. Due to the averaging process, the prediction can be error prone, however, the method is fast and efficient. As diffuse theory is assumed leading to average energy flow between compartments, the method is not suitable for low frequencies, yet this then means that the design process is tolerant to variations in geometry.

Ray tracing / ray theory or geometrical acoustics approximations are extremely useful for enclosed domains, bounded exterior spaces or rooms where diffuse theory could be applied. Each acoustic wave is represented as a ray, tracing a predictable path through space, with the incident and reflection angles from a surface taken into account. Damping can be applied at the reflective surface through a variable impedance boundary condition. As ray theory has a lower frequency limit, it is usually used for high frequency studies of room acoustics and reverberation times. The computational time varies depending on the level of complexity of the room geometry, as small changes in room angles can lead to large changes in the ray transmission paths, hence the importance of overall diffuse field assumption.

Theoretical models are used in studies to gain an understanding of the physical sound transmission processes that can occur. By idealising a geometry using simple geometrical primitives or through using significant assumptions, it may be possible to reduce the complexity of the problem down to a level where a simple theoretical model can provide insight into the problem, order of magnitude calculations and trend analysis in a parametric study. As these models are often much simpler than reality, they often are far less computationally expensive or time consuming, making them a popular first choice for initial investigations.

The boundary element is closely related to the finite element method and exploits the principle that in acoustic predictions, it is often the case that the governing transmission medium is non-dispersive and isotropic. The method shares some similarities to the finite element method in that it can utilise information from a computer aided engineering process, however the key advantage is that rather than discretising and meshing the volume, only the surfaces need treating. This leads to a reduction in computational effort and memory for a given problem. As the surfaces only need meshing, it is relatively easy to join dissimilar domains through common boundary conditions. The boundary element method requires the use of a fundamental solution governing the propagation of sound in free space, applied as a weighting function to every node on the surface mesh. This fundamental solution takes the form of a Green's function found in theoretical analysis. This usage of ideal propagation solutions provides the potential for extremely accurate solutions.

IV. NOMENCLATURE

GREEK NOTATION	EXPLANATION
$\boldsymbol{\sigma}$	VECTOR OF SOURCE AMPLITUDES OF THE FUNDAMENTAL SOLUTIONS
Γ	SURFACE
ε	ERROR
ζ	NON-DIMENSIONAL COORDINATE
λ	WAVELENGTH
μ	UNKNOWN AMPLITUDE OF THE FLUXES OF THE FUNDAMENTAL SOLUTIONS
ρ	DENSITY
σ	UNKNOWN AMPLITUDE OF THE FUNDAMENTAL SOLUTION
$\omega = 2\pi F$	FREQUENCY [RAD/SEC]
Ω	DOMAIN

Table 3 Notation for Greek notation

ENGLISH NOTATION	EXPLANATION
A	MATRIX OF BOUNDARY INTEGRALS FOR EACH SOURCE ELEMENT TO EACH OBSERVER ELEMENT
BEM	BOUNDARY ELEMENT METHOD
CAD	COMPUTER AIDED DESIGN
CAE	COMPUTER AIDED ENGINEERING
C++	COMPILER LANGUAGE
F	VECTOR OF PRESCRIBED BOUNDARY CONDITIONS FOR EACH ELEMENT
FE	FINITE ELEMENT
IREF	REFERENCE INTENSITY
PREF	REFERENCE PRESSURE
WREF	REFERENCE POWER
c_i	CONSTANT
c	SPEED OF SOUND
E	ELEMENT NUMBER
f	FREQUENCY [HZ]
G_{ij}	
H_{ij}	
$H_0^{(1)}(z)$	HANKEL FUNCTION
i	IMAGINARY NUMBER
J	JACOBIAN
k	WAVENUMBER
N	NODE NUMBER
$q = \partial u / \partial n$	FLUX OF PRESSURE
r, R	DISTANCE FROM SOURCE ELEMENT TO OBSERVER ELEMENT
u	VARIABLE (PRESSURE)
u^*	FUNDAMENTAL SOLUTION (PRESSURE)
$\frac{\partial u^*}{\partial n}$	FUNDAMENTAL SOLUTION (NORMAL PRESSURE GRADIENT)
w	WEIGHTING FACTOR
(x, y, z)	COORDINATES IN CARTESIAN SPACE

Table 4 Notation for English notation

V. KEY TERMINOLOGY

Acoustic: The medium in which a sound wave may propagate is an acoustic medium. A problem relating to sound (pressure perturbations in the audible frequency range) is considered an acoustic problem. Typically, this medium cannot sustain shear waves, only longitudinal waves (compression).

Speed of sound: It is assumed that in the elastic acoustic medium, the distance travelled by a sound wave in a given time does not vary, this is the speed of sound. For dry air at a temperature of twenty degrees Centigrade, this equates to approximately 343m/s.

Surface: The surface is a boundary between interior and exterior solution domains. The surface must be a continuous function in both displacement and slope (the method can fail should sharp corners exist in the geometry).

Node: The surface is discretised into a series of evenly spaced points. Each point on the surface is defined by three coordinates (x, y, z) where the join between nodes are assumed to be a linear segment. It is important that when discretising the surface, the nodes are spaced evenly across the whole domain, with the spacing dictated by the maximum excitation frequency of interest. Thus for exterior calculations, there should be at least four nodes per acoustic wavelength to avoid aliasing, while for interior domains the recommended minimum number rises to eight nodes per wavelength.

The spacing between nodes does not need to be the same, only that the macro-spacing is similar.

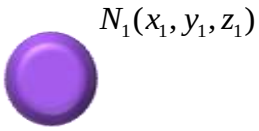

Figure 1 A node characterised by three independent coordinates

Wavelength: The acoustic frequency, f is related to the acoustic wavelength λ by the speed of sound $c = f\lambda$. It is assumed that the acoustic medium is an isotropic material where shear forces cannot be sustained.

Element: The nodes are just points in Cartesian space, however, by joining three or more unique nodes together a linear flat surface is created. This non-parabolic surface can be defined by three distinct nodes for a triangular element or four for a quadrilateral element. Through a regular distribution of node points, the elements are evenly distributed and spaced over the entire surface of the geometry. Elements may share common nodes, and it is important that in the definition of the nodes creating all elements that the normal directions are consistent. This is shown on the figure as the node connections all pointing clockwise or anticlockwise.

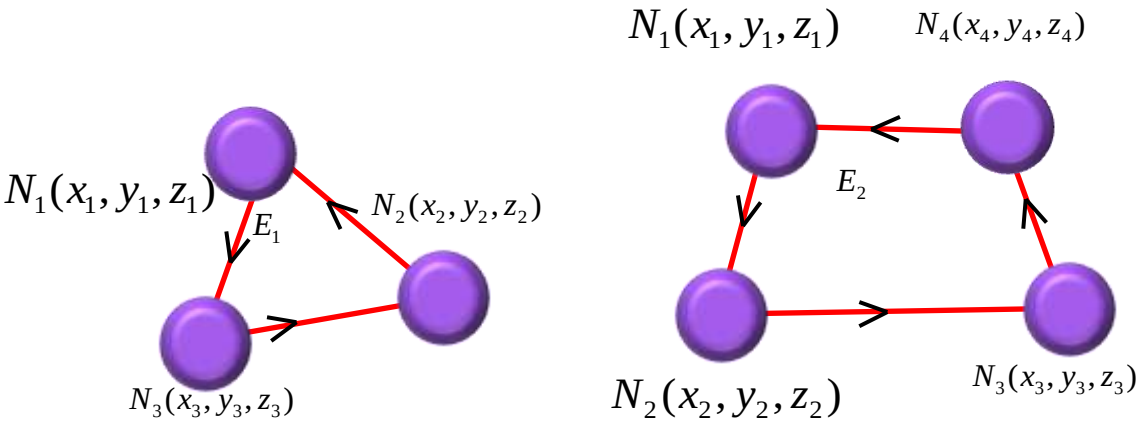

Figure 2 A triangular element characterised by three independent nodes, and a quadrilateral element characterized by four. Note that the direction normal of all elements must be consistent.

The density of the mesh is equivalent to the number of elements for a given surface area. The mesh should be smooth and the density approximately constant.

Excitation frequency: In order that a non-zero sound pressure field is calculated, the non-trivial solution, the geometric domain must contain at least one sound source, otherwise the only sound field that can exist is silence. This monopole source is located at an arbitrary position. The boundary element calculation routine calculates a predicted sound field for each discrete excitation frequency, up to a finite upper limit. There are particular numerical problems when attempting to calculate the steady state acoustic pressure deflection (the frequency equal to zero Hertz 0Hz). In order for the calculation to converge to a satisfactory solution, it is required that the zero frequency instead be replaced by a relatively small value, for example 0.01Hz.

Boundary condition: Each element has an associated boundary condition which is an initial condition required to be able to solve the problem. The types of physical condition which can be applied to the element include a prescribed velocity or pressure, including complex impedance. These are proportional to the fundamental solution.

In the input file, each element has an associated boundary condition. In the code, these are denoted boundary condition types 1 or 2 or 3.

The pressure on an observer element can be written in terms of a fundamental solution on a source element a distance r away. This is a type 1 boundary condition (pressure).

As will be described in the following sections, the velocity applied to the element (type 2) can be written in terms of a fundamental solution, as can the pressure, from this observer element to a source element a distance r away. The normal velocity can be applied to the element to represent a solid surface, or prescribed flow condition. The amplitude of the fundamental solution provided by the source element yielding the appropriate boundary condition on the observer element is denoted Q.

$$p = i\rho\omega Q u^*$$
$$V = -Q\left(\frac{1}{r} + \frac{i\omega}{c}\right)u^*$$

{Note: Determine if it is normal velocity only or magnitude of velocity which is zero. Difference is rx/|r| type term but in the unit normal direction of the element.}

The alternative boundary condition is a (type 3) impedance condition, where the complex value of normal pressure to normal velocity is specified. Although this can also be written in terms of a fundamental solution, it may be shown that these terms cancel.

$$Z = p/V$$

Fundamental solution: The boundary element method relies on the use of fundamental solutions to be applied to each node, as a weighting factor. In the case of acoustic predictions, the fundamental solutions take the form of the Green's function solution to the Helmholtz equation. This represents the propagation of a wave through free space and can be applied in either two-dimensional or three-dimensional forms (the two-dimensional form utilises Hankel functions rather than exponential functions). Further details are provided in the section on monopole sound fields.

HELMHOLTZ FUNDAMENTAL SOLUTIONS (MONOPOLES)		
	u^*	$\frac{\partial u^*}{\partial n}$
2D	$\frac{1}{4i} H_0^{(2)}(kr)$	$-\frac{k}{4i} H_1^{(2)}(kr)$
3D	$\frac{1}{4\pi r}\exp(-ikr)$	$\frac{1}{4\pi r}\left(-\frac{1}{r}-ik\right)\exp(-ikr)$

Table 5 Fundamental Solutions for acoustic wave propagation in free space

These Green's functions represent how a wave would propagate through the free space, between the source position and observer location a distance r away. It is important to understand that the fundamental solution can be directly linked to the parameters of interest for acousticians, such as fluid velocity or pressure.

To take an example, a sphere of radius a pulsating with a frequency $\omega = 2\pi F$ and a radial velocity $v(a)$ at the surface generates a spherical wave. The oscillatory volume flow at the surface has an amplitude Q that is given by the monopole strength $Q = 4\pi a^2 v(a)$. In the limiting case of a monopole point source, the wavelength dimension $ka << 1$ we obtain for the acoustic pressure $p(r) = \left(1/4\pi R\right)(i\rho\omega)Q\exp(-ikR)$. In the code listed below, the source strength is user provided as a complex constant B.

Therefore, both the velocity V and the pressure p can be written in terms of this fundamental solution.

$$p = i\rho\omega Q u^*$$

$$V = Q\frac{\partial u^*}{dr}$$

The intensity can also be related to the acoustic pressure and velocity through the analogy with the power dissipation of electrical circuits.

$I = \frac{1}{2}\mathrm{Re}\{p\bar{V}\}$, where the overbar denotes the complex conjugate is taken.

Data recovery mesh: The boundary element method starts with a known boundary condition on a geometrical surface, and the unknown distribution of fundamental solutions across the domain which yield that boundary condition is formulated in terms of a matrix problem. Once this distribution is known, the sound pressure can be obtained everywhere in the domain. Data recovery points are the locations where the user is interested in knowing these resulting sound pressures, intensities or acoustic fluid velocities. Although it is described as a mesh for graphical interpretive purposes, it can equally be defined as a set of single unconnected points. In designing the data recovery mesh or locations of the recovery points, it may be of interest to obtain the acoustic pressure close to the solid geometry. In order to avoid singularities in the matrix solution, it is necessary to locate each data recovery point at least 1/4 of an acoustic wavelength away from the solid geometry element or node.

The sound field from multiple monopoles: A monopole is defined as a solution to the Helmholtz equation which yields a sound field which propagates omni-directionally. For a three-dimensional field, this results in an omni-directional field where the amplitude falls away as a function of the inverse of the radius. This is illustrated in Figure 3 for a monopole originating on the left hand side of the screen and shown propagating to the right hand side of the diagram.

When several monopoles are situated in a domain, it is possible to obtain very complicated sound fields through linear superposition leading to regions of high sound pressures from constructive interference and regions of low sound pressure from waves destructively interfering. This is most easily illustrated using a large number of monopoles situated close to each other, in a baffled piston arrangement. At lower frequencies, as seen in Figure 4, the monopoles are spaced within an acoustic wavelength of each other such that the overall sound field is omni-directional. However at higher frequencies, as also shown in Figure 4, the spacing between each monopole becomes more significant compared to the acoustic wavelength and the phase change between each monopole results in a more complicated sound field, with lobe patterns. This is the primary reason why loudspeaker packages comprise many different speaker sizes, each providing sound in a specified frequency range. In the boundary element method, we assume that we know the boundary conditions applied to the surface of a geometry and require knowledge of the amplitude and phase of a distribution of fundamental solutions on each node that satisfies this boundary condition.

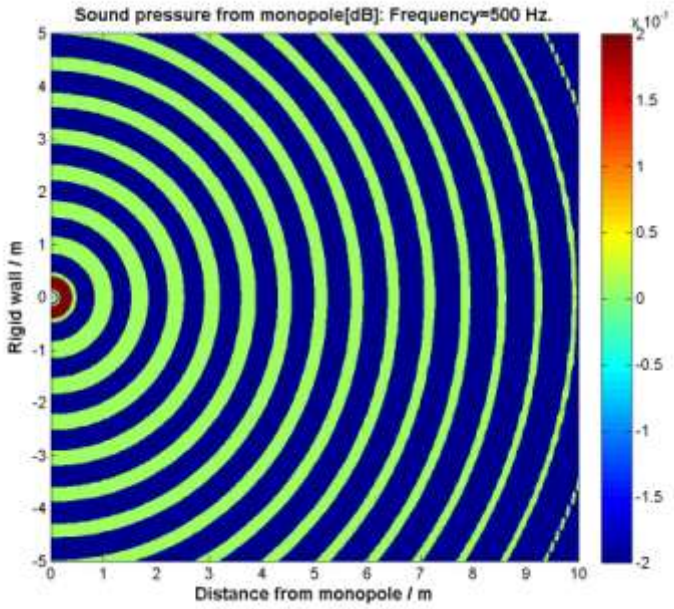

Figure 3 Omni-directional sound pressure propagating from a monopole fundamental solution.

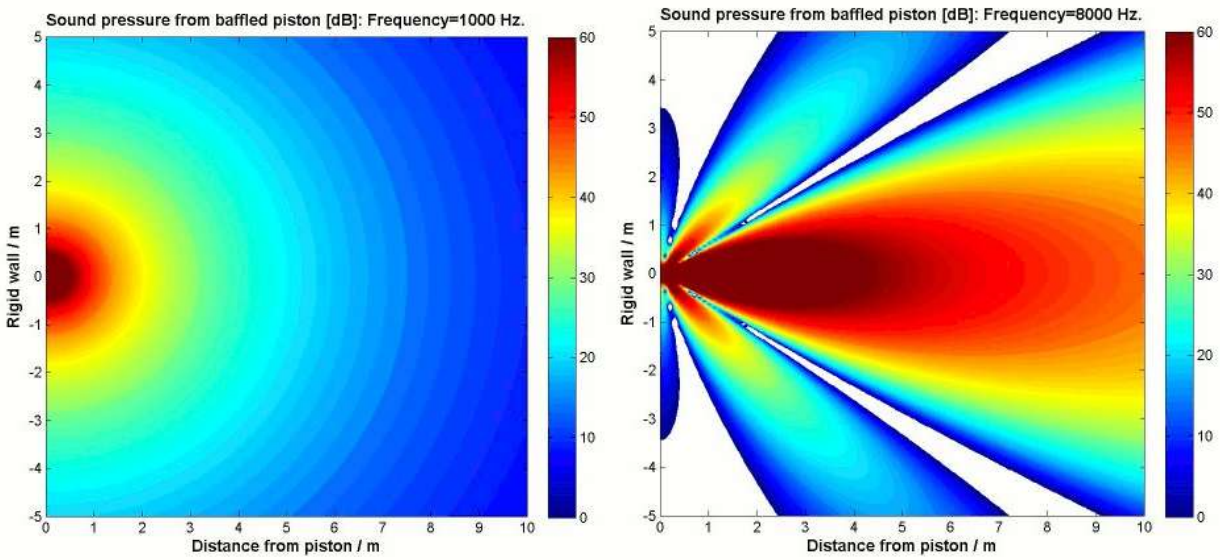

Figure 4 Directional sound field propagating from a distribution of monopoles at low and high frequencies.

VI. MATHEMATICAL FOUNDATION OF THE BOUNDARY ELEMENT METHOD

A typical boundary element code has several advantages over traditional finite differencing calculation methods. As with finite element codes, a geometry is described and a mesh generated, however, only the bounded surfaces require meshing, which allows not just a speed advantage, but also the possibility to investigate domains which are relatively large in comparison to the geometry being examined.

Broadly speaking, boundary conditions are applied to the surface under investigation, and solutions found for the amplitude of potentials applied to the surface which yield those boundary conditions. Once these amplitudes are known, they may be used to obtain the displacement, velocity or other parameter inside the domain.

The aim of this section is to provide an illustration of the boundary element method, through the work of Brebbia and Walker. [C. A. Brebbia, "The Boundary Element Method for Engineers". Pentech Press, 1980. C. A. Brebbia and S. Walker, "Boundary Element Techniques in Engineering, Newnes-Butterworths, 1980].

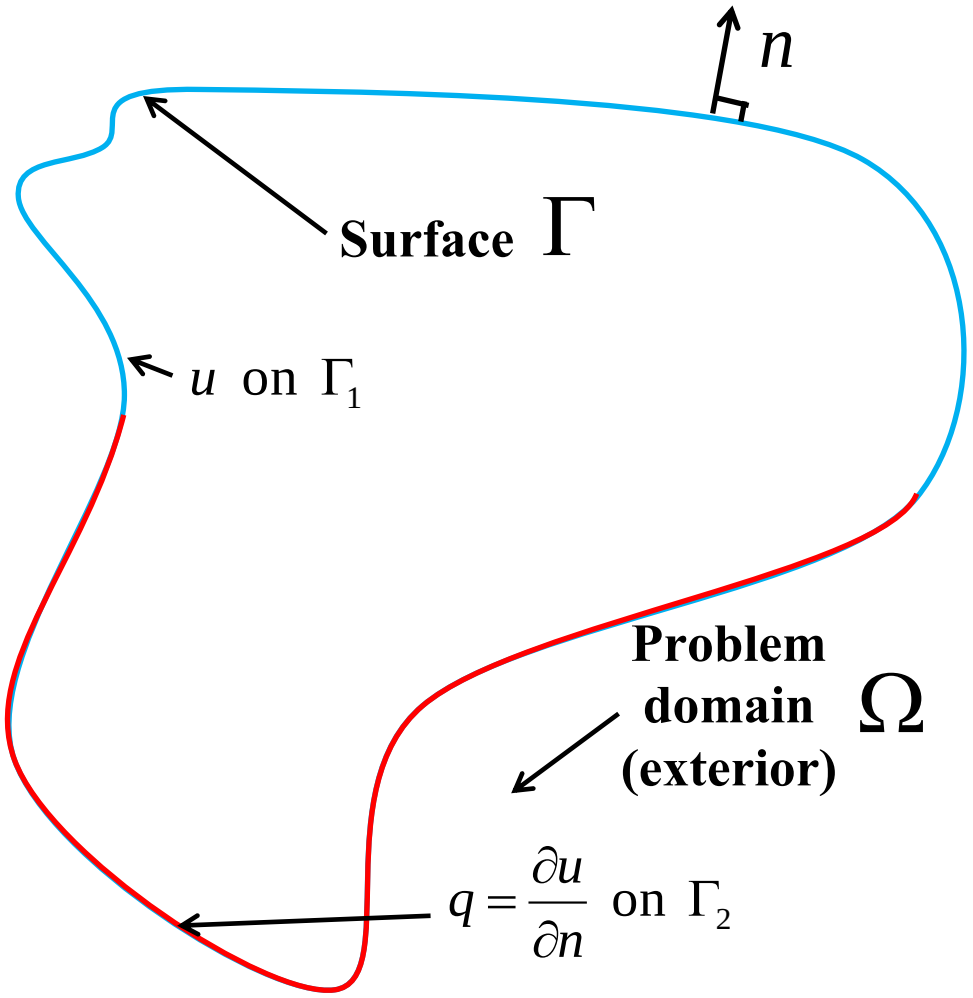

Figure 5 Definition of a domain for acoustic calculations.

A domain is defined by the symbol Ω and a surface which is closed inside it Γ. This surface is separated into two sections, joined together to create a closed surface, with the first section denoted Γ_1 and the second Γ_2 as shown in Figure 5. On each surface are located surface potentials of unknown amplitude which give rise to certain physical conditions on the surface.

On the first, Γ_1, for the acoustic calculation is a term u while in on the surface Γ_2 we define the property of interest to be the normal derivative, the flux $q = \partial u / \partial n$. It is possible to write the physical boundary conditions of pressure or velocity in terms of the amplitudes of the fundamental solutions.

By converting physical conditions such as pressure or velocity into just the amplitudes of the fundamental solutions, the problem can be decomposed into a generic solution to a matrix.

The exact solutions for the physical properties are denoted by the functions $\bar{u}$ and $\bar{q}$ on the sections of the surface respectively. This allows an error on the surfaces to be defined as,

$\varepsilon_1 = u - \bar{u}$	on Γ_1
$\varepsilon_2 = q - \bar{q}$	on Γ_2
$\Gamma = \Gamma_1 + \Gamma_2$	

The aim of the boundary element method is to determine the amplitude of a distribution of potentials so as to minimise the error. This error is distributed over the surface according to a weighting function w, which may be a fundamental solution to the governing equation in the domain.

The distribution of the error in the domain is written,

$$\int_{\Omega} \varepsilon w d\Omega = \int_{\Gamma_2} \varepsilon_2 w d\Gamma - \int_{\Gamma_1} \varepsilon_1 \frac{\partial w}{\partial n} d\Gamma .$$

For the Helmholtz problem, the governing equation is $\nabla^2 u + \lambda^2 u = 0$, where the wavelength is denoted λ. Substitution of the error residuals and weightings (fundamental solutions) provides the starting weighting residual statement.

$$\int_{\Omega} \left(\nabla^2 u + \lambda^2 u\right) u^* d\Omega = \int_{\Gamma_2} \left(q - \bar{q}\right) u^* d\Gamma - \int_{\Gamma_1} \left(u - \bar{u}\right) \frac{\partial u^*}{\partial n} d\Gamma$$

where u^* describes the fundamental solution to the Helmholtz equation, $\nabla^2 u^* + \lambda^2 u^* = -\delta_i$ (also termed the Green's function).

The first term may be integrated by parts,

$$\int_{\Omega} \left(\nabla^2 u^* + \lambda^2 u^*\right) u d\Omega = \int_{\Gamma_2} \left(q - \bar{q}\right) u^* d\Gamma - \int_{\Gamma_1} \left(u - \bar{u}\right) \frac{\partial u^*}{\partial n} d\Gamma - \int_{\Gamma} q u^* d\Gamma + \int_{\Gamma} u q^* d\Gamma$$

It is noted that we have specified that the boundary conditions on the two parts of the surface must exactly meet the exact solution. Therefore, this simplifies the equation to,

$$\int_{\Omega} \left(\nabla^2 u^* + \lambda^2 u^*\right) u d\Omega = -\int_{\Gamma} q u^* d\Gamma + \int_{\Gamma} u q^* d\Gamma$$

The integration over the domain of the fundamental solution yields only a delta term, located at the point of interest "i".

$$-u_i = -\int_{\Gamma} q u^* d\Gamma + \int_{\Gamma} u q^* d\Gamma$$

This final equation relates the value of u at the location of point "i" with the values of u and q over the domain Γ and it applies when the point is inside the interior domain Ω. The advantage of boundary integrals appears when we only consider points on the surface of the geometry, therefore the point needs to be moved to this boundary. Due to the singularity which occurs at this point, we analytically represent this through a constant c. For a point inside the boundary, $c_i = 1$, for a point on the boundary it is 0.5 and outside the boundary it is zero.

$$c_i u_i + \int_{\Gamma} q^* u d\Gamma = \int_{\Gamma} q u^* d\Gamma$$

This is the governing boundary element equation for use with a fundamental Helmholtz solution. Such solutions for the fundamental equation are shown in Table 5.

The boundary element equation is a continuous integral, however the numerical implementation is a discrete equation with truncation errors through the continuous domain being represented by a series of linear joins, as shown in Figure 6. The requirements for this discretisation, the type of elements which can be created and the number of elements for a given acoustic frequency are discussed later in this manual.

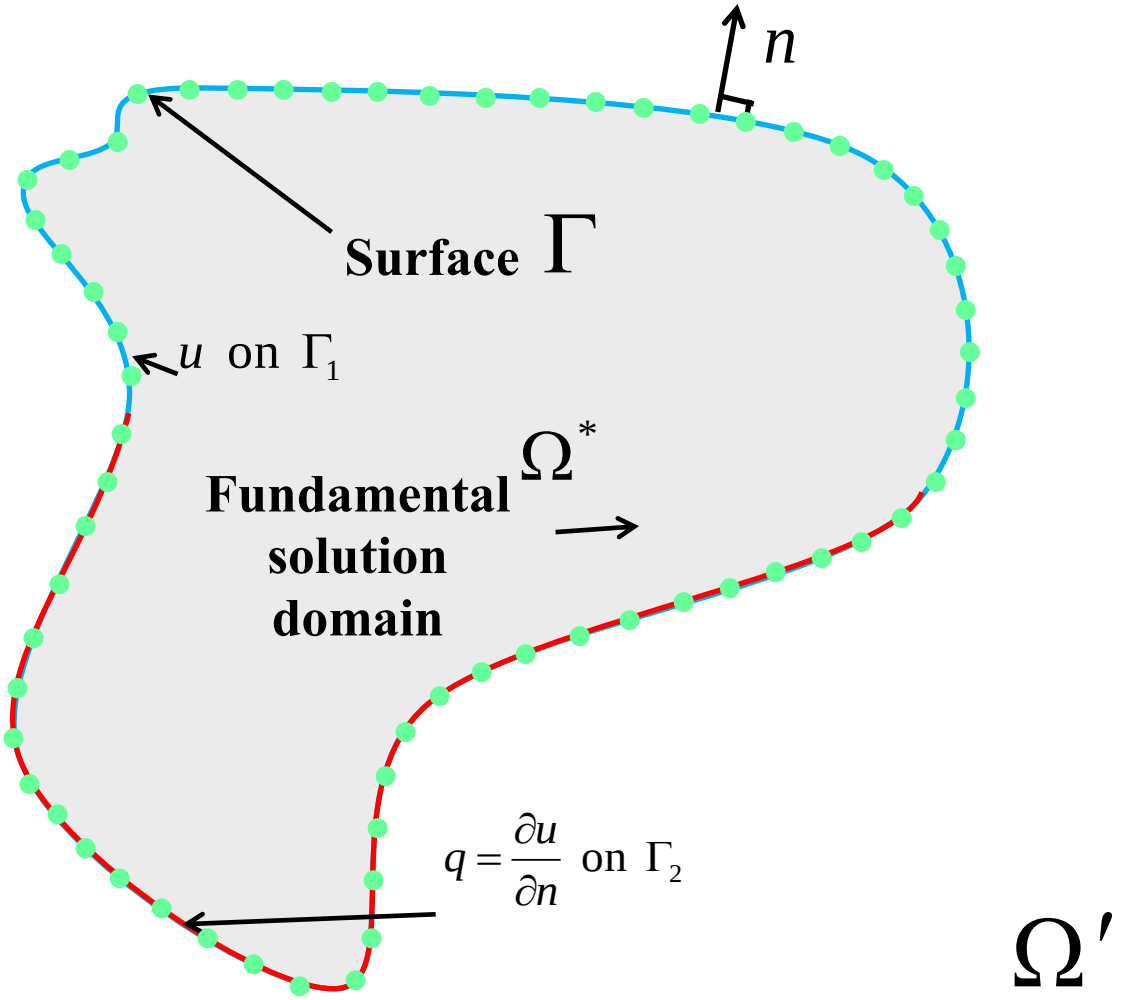

Figure 6 Discretisation of the surface bounding the solution domain. The surface is created by joining nodes together with linear segments. When completed over a three dimensional domain, the linear segments create surface areas (elements).

PRINCIPLE OF THE INDIRECT BOUNDARY ELEMENT METHOD

The formulation of the indirect boundary element method starts with the boundary integral equation $c_i u_i + \int_\Gamma q^* u d\Gamma = \int_\Gamma q u^* d\Gamma$ applied to an interior domain Ω.The indirect method starts with a solution which satisfies the governing equation in the domain but has unknown coefficients at a number of points. At this point, we also introduce to the formulation a monopole source located inside the domain which will allow the non-trivial solution to be obtained (without this, the only sound field that exists is silence). The amplitude of this source is b with a location inside Ω.

$$c_i u_i + \int_\Gamma q^* u d\Gamma = \int_\Gamma q u^* d\Gamma - \int_\Omega b u^* d\Omega$$

As shown in Figure 6, the surface can either be integrated as a continuous surface (we will use the continuous form for the present section) or discretised by nodes joined with linear elements.

In order to enforce the boundary condition on the surface, it becomes necessary to also consider a domain exterior to the surface Ω' where no sources exist and the properties u', q' are governed by the equation $\int_\Gamma q^* u' d\Gamma + \int_\Gamma q' u^* d\Gamma = 0$ (as the surface is pointing in the opposite direction, the sign is reversed).

As part of the derivation, we assume that the solution generated for the exterior field is identical to that generated for the interior field on the surface. Therefore we are able to write,

$$c_i u_i + \int_\Gamma q^* u d\Gamma - \int_\Gamma q u^* d\Gamma + \int_\Omega b u^* d\Omega = \int_\Gamma q^* u' d\Gamma + \int_\Gamma q' u^* d\Gamma$$

On the surface, it is possible to write that $u = u'$ as exact terms. Rearranging the above equation, these terms cancel with each other.

$$c_i u_i + \int_\Gamma q^* (u - u') d\Gamma - \int_\Gamma (q u^* + u^* q') d\Gamma + \int_\Omega b u^* d\Omega = 0$$

$$c_i u_i - \int_\Gamma u^* (q + q') d\Gamma + \int_\Omega b u^* d\Omega = 0$$

If we examine the form of this equation, we can define $\sigma = (q + q')$ as the unknown density of the fundamental solutions or amplitudes of the fundamental solutions across the surface of the domain, necessary to generate u_i. The constants can be taken inside this variable and summarised as,

$$u_i = \int_\Gamma u^* \sigma d\Gamma - \int_\Omega b u^* d\Omega$$

An alternative formulation is clearly gained by writing the sum of the derivatives of u, the fluxes as equal to zero. If $q + q' = 0$ then the boundary integral equation becomes

$$c_i u_i + \int_\Gamma q^* (u - u') d\Gamma - \int_\Gamma u^* (q + q') d\Gamma + \int_\Omega b u^* d\Omega = 0$$

$$c_i u_i - \int_\Gamma q^* (u - u') d\Gamma + \int_\Omega b u^* d\Omega = 0$$

Writing the dipole term $\mu = (u' - u)$ relates the value of u_i in the domain to the amplitude of a series of dipoles distributed over the whole surface.

$$u_i = \int_\Gamma q^* \mu d\Gamma - \int_\Omega b u^* d\Omega$$

The former method has greater advantages as lower orders of derivatives of the fundamental solution are utilised. If we choose this former method, then we can write u_i and the derivative, the flux, as,

$$u_i = \int_\Gamma u^* \sigma d\Gamma - \int_\Omega b u^* d\Omega, \qquad q_i = \frac{\partial u}{\partial n} = \int_\Gamma q^* \sigma d\Gamma - \int_\Omega b u^* d\Omega$$

This applies for any point on the inside of the domain, however, in order to find the value of a point on the boundary, the location i needs to be moved to the surface. A singularity exists, therefore, the integration is carried out analytically, to obtain,

$$u_i = \int_\Gamma u^* \sigma d\Gamma - \int_\Omega b u^* d\Omega, \qquad q_i = -\frac{\sigma_i}{2} + \int_\Gamma q^* \sigma d\Gamma - \int_\Omega b u^* d\Omega$$

DISCRETISATION OF THE SURFACE DOMAIN

Up until this point, the boundary integral equation has been considered as a continuous integral over the surface in the domain. There are a number of challenges with this method, most notably that few geometries exist in reality where there is simple analytical solution for the boundary integral. It is more likely that the geometry is a component in an automotive, aeronautical, aerospace or industrial environment.

For this reason, the integral has to be completed using a numerical method, which requires that the surface be discretised into a series of nominally evenly spaced nodes (see Figure 1 for a description of the node and Figure 6 to see how the surface is broken up into elements separated by these nodes).

There are N elements from $i=1$ to N, where we may describe two different elements by their indices i and j. The value of u on element i is denoted u_i while the value attached to element j is u_j. We may then begin to write the discretised formulation in matrix notation using a summation over N elements. At this point we do not consider the implications of $i = j$ in the mathematical derivation. In addition, we temporarily leave the source term B out of the equations, for brevity.

$$u_i = \sum_{j=1}^{N} \sigma_j \int_{\Gamma_j} u^* d\Gamma$$

$$q_k = -\frac{1}{2}\sigma_k + \sum_{j=1}^{N} \sigma_j \int_{\Gamma_j} q^* d\Gamma$$

The boundary integrals are provided with their own specific notation for the matrices, such that $G_{ij} = \int_{\Gamma_j} u^* d\Gamma$ and $\hat{H}_{ij} = \int_{\Gamma_j} q^* d\Gamma$.

$$u_i = \sum_{j=1}^{N} \sigma_j G_{ij}$$

$$q_k = -\frac{1}{2}\sigma_k + \sum_{j=1}^{N} \sigma_j \hat{H}_{kj}$$

Finally, for the boundary integral for the fluxes, we may write that if $i \neq j$ then $H_{ij} = \hat{H}_{ij}$, whereas for $i = j$, $Hii = \hat{H}ii - 1/2$.

$$u_i = \sum_{j=1}^{N} \sigma_j G_{ij}$$

$$q_k = \sum_{j=1}^{N} \sigma_j H_{kj}$$

Now we introduce the discretisation of the source term. These can be used to include either monopole sources or body forces. These are written as $B_i = \int_{\Omega} b u^* d\Omega$.

Our final discretised indirect boundary integral equation is left as,

$$u_i = \sum_{j=1}^{N} \sigma_j G_{ij} - B_i^u$$

$$q_k = \sum_{j=1}^{N} \sigma_j H_{kj} - B_k^u$$

In this code, the matrices are hard coded for N elements as $\mathbf{F} = \mathbf{A}\boldsymbol{\sigma}$.

$$\underbrace{\begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{Bmatrix}}_{KNOWN} = \underbrace{\begin{bmatrix} G_{11} & G_{12} & G_{13} & \\ G_{21} & G_{11} & & \\ G_{31} & & \ddots & \\ & & & G_{NN} \end{bmatrix}}_{KNOWN} \underbrace{\begin{Bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_N \end{Bmatrix}}_{UNKNOWN}$$

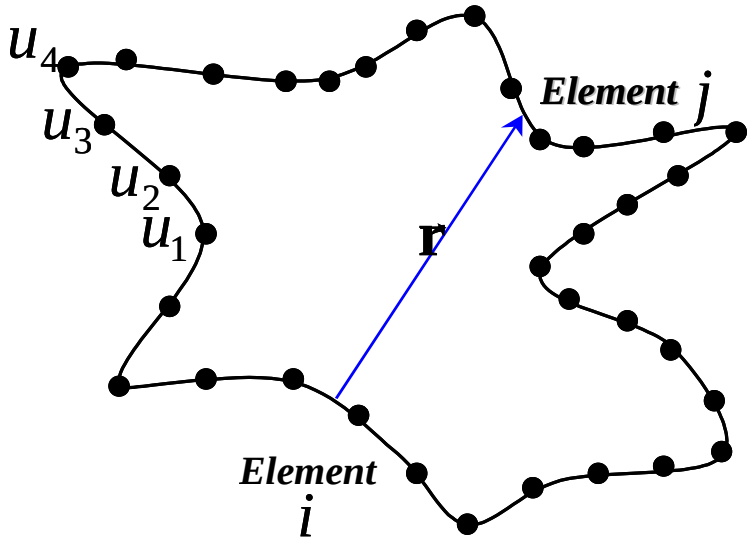

Figure 7 Boundary integral surface illustrating how each element takes into account the effect of a source of unknown amplitude located at each element. The problem is then posed as "what amplitude of all of the sources yields the correct known boundary condition at this element?" in matrix form.

The effect of each fundamental solution located at the nodes are shown in Figure 7, where it may be seen that the vector $\mathbf{r}$ is never zero, as two nodes are never co-located.

The integration of the boundary element is usually the most complex part of the boundary element code, due to the fact that the surface is generally not aligned with the principle axes of the problem (see for example the example in any of the previous figures) but rather with the normals pointing in an arbitrary direction (yet consistent with each other).

In this document, two integral methods are shown as examples, relating to the integration of a quadrilateral and also triangular element respectively. Both element types are calculated in a similar manner.

The components of the matrix $\mathbf{A}[i,j] = G_{ij} = \int_{\Gamma_j} u^* d\Gamma_j$, where the fundamental solution is assumed to be o the form $u[i] = \frac{1}{4\pi r}\exp(-i\omega r/c)$ where the frequency in radians / second is related to the frequency in Hz by $\omega = 2\pi F$ and r is the distance between the source element and the observer element. The element is defined using non-dimensional coordinates and an interpolation routine is used to obtain the values of the fundamental solution across the whole element. A linear integration method is then applied to determine the value of the integration over the area of the element.

QUADRILATERAL ELEMENTS

The principle axes are (x, y, z), however the linear element may be sloping in directions which are different to this. Therefore there needs to be an integration along non-dimensional coordinates. This involves a change of variable in the double integral using a Jacobian method.

The change of variable in a double integral using a Jacobian can be summarised from Kreyszig as,

$$\int_a^b f(x)dx = \int_a^b f(x(u))\frac{\partial x}{\partial u}du$$

$$= \int_{\zeta_1}\int_{\zeta_2} f\left(x(\zeta_1,\zeta_2), y(\zeta_1,\zeta_2)\right)\left|\frac{\partial(\zeta_1,\zeta_2)}{\partial(\zeta_1,\zeta_2)}\right| d\zeta_1 d\zeta_2$$

$$|J| = \begin{vmatrix} \frac{\partial x}{\partial \zeta_1} & \frac{\partial x}{\partial \zeta_2} \\ \frac{\partial y}{\partial \zeta_1} & \frac{\partial y}{\partial \zeta_2} \end{vmatrix}$$

$$G_{ij} = \int_{\Gamma_j} u^* d\Gamma_j$$

$$= \int_{\zeta_1}\int_{\zeta_2} u^* |G| d\zeta_1 d\zeta_2$$

$$|G| = \sqrt{\left(g_1^2 + g_2^2 + g_3^2\right)}$$

We now look at the element j, with four labels denoting nodes k=1, 2, 3 and 4.

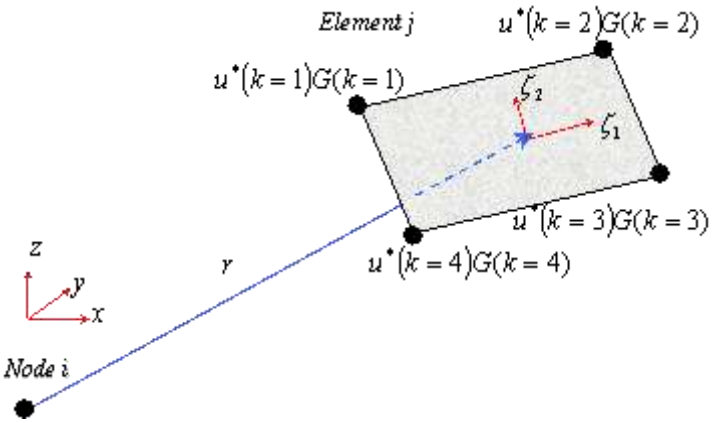

Figure 8 Quadrilateral boundary integral surface showing non-dimensional coordinate form of the surface direction.

For each coordinate node on the element, the product $u^* |G|$ is obtained, where the surface directions are obtained through an interpolation routine (the details are provided in the coding explanation).

$$g_1 = \left(\frac{\partial y}{\partial \zeta_1}\frac{\partial z}{\partial \zeta_2} - \frac{\partial y}{\partial \zeta_2}\frac{\partial z}{\partial \zeta_1} \right)$$

$$g_2 = \left(\frac{\partial z}{\partial \zeta_1}\frac{\partial x}{\partial \zeta_2} - \frac{\partial x}{\partial \zeta_1}\frac{\partial z}{\partial \zeta_2} \right)$$

$$g_3 = \left(\frac{\partial x}{\partial \zeta_1}\frac{\partial y}{\partial \zeta_2} - \frac{\partial y}{\partial \zeta_1}\frac{\partial x}{\partial \zeta_2} \right)$$

Once the product of $u^* |G|$ is found at each of the corners, the numerical integration can take place. For a linear surface, the following is appropriate.

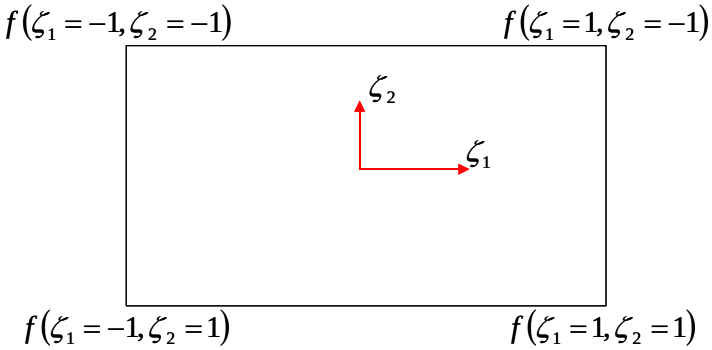

Figure 9 Quadrilateral linear integration method.

$$\int_{-1}^{1}\int_{-1}^{1} f(\zeta_1,\zeta_2)\,d\zeta_1 d\zeta_2 = \sum_{m=1}^{2}\sum_{l=1}^{2} w_l w_m f(\zeta_m,\zeta_l)$$

l,m	$\zeta_{l,m}$	$w_{i,m}$
1	$1/\sqrt{3}$	1
2	$-1/\sqrt{3}$	1

This is Gauss' integration method for linear elements in two dimensions (non-dimensionalised), See Abramowitz and Stegun P887, P916 for weightings .

So $\int_{-1}^{1}\int_{-1}^{1} f(\zeta_1,\zeta_2)d\zeta_1 d\zeta_2 = f\left(-1/\sqrt{3},-1/\sqrt{3}\right)+ f\left(-1/\sqrt{3},1/\sqrt{3}\right)+ f\left(1/\sqrt{3},-1/\sqrt{3}\right)+ f\left(1/\sqrt{3},1/\sqrt{3}\right)$, where $f(\zeta_1,\zeta_2)=u^*G(\zeta_1,\zeta_2)$. This is the final boundary integral for the linear quadrilateral element.

TRIANGULAR ELEMENTS

The process is similar for triangular elements, first the value of the fundamental solution is obtained between each node i and nodes which make up the three corners of the triangular element j, k=1,2,3.

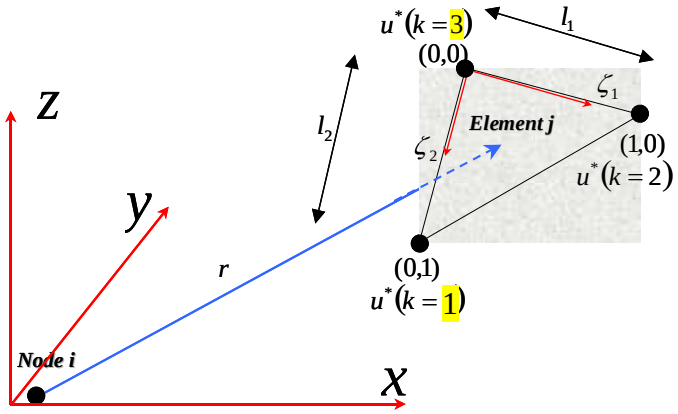

Figure 10 Triangular linear integration method.

The position of a point on the triangular element is,

$$r = x\vec{i} + y\vec{j} + z\vec{k} = x_2\vec{i} + y_2\vec{j} + z_2\vec{k} + l_1\zeta_1\vec{e}_1 + l_2\zeta_2\vec{e}_2$$

$$\vec{e}_1 = \left(\frac{x_1 - x_3}{l_1}\right)\vec{i} + \left(\frac{y_1 - y_3}{l_1}\right)\vec{j} + \left(\frac{z_1 - z_3}{l_1}\right)\vec{k}$$

$$\vec{e}_2 = \left(\frac{x_2 - x_3}{l_2}\right)\vec{i} + \left(\frac{y_2 - y_3}{l_2}\right)\vec{j} + \left(\frac{z_2 - z_3}{l_2}\right)\vec{k}$$

These are rearranged to obtain any point on the triangular element as a function of non-dimensional coordinates.

$$x = \zeta_1 x_1 + \zeta_2 x_2 + (1-\zeta_1-\zeta_2)x_3$$
$$y = \zeta_1 y_1 + \zeta_2 y_2 + (1-\zeta_1-\zeta_2)y_3$$
$$x = \zeta_1 z_1 + \zeta_2 z_2 + (1-\zeta_1-\zeta_2)z_3$$
$$\zeta_3 = 1-\zeta_1-\zeta_2$$
$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix} \begin{Bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \end{Bmatrix}$$

$$u = u_1\zeta_1 + u_2\zeta_2 + u_3\zeta_3 = \sum_{l=1}^{3} u_l\zeta_l$$

Once the values at an arbitrary location on the triangular element are known, the integration over the surface area can be found using a simple Gaussian integration method for triangular elements, where the non-dimensional coordinate varies from 0 to 1.

$$G_{ij} = \int_{\Gamma_j} u^* d\Gamma_j$$

$$= \int_{\zeta_1} \int_{\zeta_2} u^* \mid G \mid d\zeta_1 d\zeta_2$$

$$\mid G \mid = \sqrt{\left(g_1^2 + g_2^2 + g_3^2\right)}$$

$$g_1 = \left(\frac{\partial y}{\partial \zeta_1}\frac{\partial z}{\partial \zeta_2} - \frac{\partial y}{\partial \zeta_2}\frac{\partial z}{\partial \zeta_1} \right)$$

$$g_2 = \left(\frac{\partial z}{\partial \zeta_1}\frac{\partial x}{\partial \zeta_2} - \frac{\partial x}{\partial \zeta_1}\frac{\partial z}{\partial \zeta_2} \right)$$

$$g_3 = \left(\frac{\partial x}{\partial \zeta_1}\frac{\partial y}{\partial \zeta_2} - \frac{\partial y}{\partial \zeta_1}\frac{\partial x}{\partial \zeta_2} \right)$$

$$\int_{-1}^{1}\int_{-1}^{1} f(\zeta_1,\zeta_2) d\zeta_1 d\zeta_2 = \sum_{m=1}^{2}\sum_{l=1}^{2} w_l w_m f(\zeta_m,\zeta_l)$$

l,m	$\zeta_{l,m}$	$w_{i,m}$
1	2/3	1/3
2	1/6	1/3

This is Gauss' integration method for linear elements in two dimensions (non-dimensionalised), See Abramowitz and Stegun P887, P916 for weightings .

So $\int_{-1}^{1}\int_{-1}^{1} f(\zeta_1,\zeta_2) d\zeta_1 d\zeta_2 = \frac{1}{3} f(2/3,1/6) + \frac{1}{3} f(1/6,1/6) + \frac{1}{3} f(1/6,1/3)$, where $f(\zeta_1,\zeta_2) = u^* G(\zeta_1,\zeta_2)$. This is the final boundary integral for the linear triangular element.

SOURCE DEFINITION AND LOCATION

For a non-trivial solution, the domain needs to include at least one source, with defined amplitude Q and location. If the distance between the source and the observer element is given by Rs, then the pressure and velocity may be written in terms of this distance and the fundamental solution.

$$u^* = \frac{1}{4\pi R_S} \exp(-i\omega R_S / c)$$

$$p = iQ\rho\omega u^*$$

$$V = Q\frac{\partial u^*}{\partial r}$$

DATA RECOVERY PARAMETERS

The data recovery points are incorporated into the input file as a list of points where the user is interested in knowing the acoustic pressure, intensity or velocity. They must be at least one quarter of an acoustic

wavelength away from the solid geometry (boundary) in order to avoid singularities occurring in the boundary integral formulation.

The data recovery points can either be designed as single points or through the use of a meshing program, which allows a fuller graphical representation of the end data.

The preferred format for the data recovery mesh is through the use of a Patran Neutral file with extension "*.out.1*". Although Patran is a commercial program, many converters exist for moving between one neutral format to another.

As a result, it is relatively easy to obtain the data recovery mesh in this appropriate format, however, it is also possible to directly program typical geometries into the correct surface mesh format through the use of a bespoke program. One such example is provided here, in the subdirectory "*DRMGen*".

The Patran file format contains information about the location of the nodes making up the data recovery mesh, and also how they are connected to create a structured orderly design for graphical processing. A file may also include data on material properties and node loading, however these are not of interest for this application.

The scripted mesh generator provides a structured grid of quadrilateral elements with a fixed number in the x and y directions. It models a mesh surrounding an automotive tyre, see Figure 15 for an illustration. The first few lines of the file describe how many nodes are to be defined in the program and a date (and time) of creation.

The nodal data number is defined, followed by it's position in three dimensional Cartesian coordinates, x,y,z. Following this, a list of the elements are defined. These quadrilateral elements connect four nodes together. As previously described, it is vital that the normal directions of conjoining elements be consistent.

The C++ data recovery mesh program can be compiled using the command *"g++ DRMGen_Tyre.cpp –o a.out".*

Once the location of the data recovery nodes are known it remains to determine the physical parameters of interest. We start with the equation for the known boundary condition relating to the matrix of boundary integrals and their associated amplitudes $\mathbf{F} = \mathbf{A}\boldsymbol{\sigma}$. At this point we assume that we have a suitable solution for the amplitudes $\boldsymbol{\sigma}$, which will allow all physical parameters to be determined inside the domain.

If the amplitudes of the fundamental solutions are known across the surface, as shown in Figure 11, then the corresponding boundary integrals can be calculated between each data recovery node and each boundary element on the surface. This is written as, $\mathbf{F}_{DRM} = \mathbf{A}_{DRM}\boldsymbol{\sigma}$.

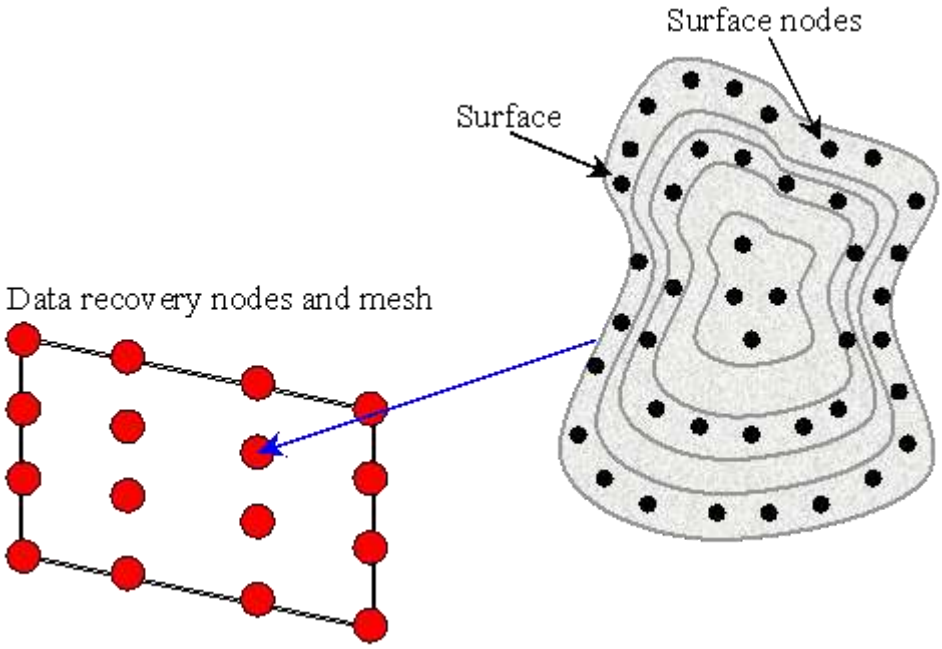

Figure 11 Boundary integrals from the surface to the data recovery mesh.

In order to obtain the pressure and velocity at each data recovery point, the relationship between the fundamental solution and these physical relations are re-stated.

$$u^* = \frac{1}{4\pi r}\exp(-i\omega r/c)$$

$$p = iQ\rho\omega u^*$$

$$V = Q\frac{\partial u^*}{\partial r}$$

The pressure may be calculated at each of the data recovery points thus, $\mathbf{P}_{DRM} = i\rho\omega\mathbf{F}_{DRM}$ as the amplitude of the source (usually Q) is given by σ. In addition to this, the contribution from the source location is also added.

The output variables relating to the sound pressure are denoted for the i'th node: Outputvar[i][0] to Outputvar[i][2].

Outputvar[i][0] Sound pressure in dB re Pref.

$$= 20\log 10\left(\frac{\sqrt{\mathrm{Re}(P_{DRM})^2 + \mathrm{Im}(P_{DRM})^2}}{P_{\mathrm{REF}}\sqrt{2}}\right)$$

Outputvar[i][1] Sound pressure magnitude [N/m^2].

$$= \sqrt{\mathrm{Re}(P_{DRM})^2 + \mathrm{Im}(P_{DRM})^2}$$

Outputvar[i][2] Sound pressure phase in degrees.

$$= 180\left(\arctan\left[\frac{\mathrm{Im}(P_{DRM})}{\mathrm{Re}(P_{DRM})}\right]\right)/\pi$$

The velocity is given by $V = -Q\{i\omega/c + 1/r\}u^*$ where r is the distance from the source element to the observer data recovery node. The velocity is different from the pressure in that the former is a scalar whereas the velocity can be defined in terms of components in the three Cartesian axes.

$$V_x = V\frac{r_x}{|r|}, \qquad V_y = V\frac{r_y}{|r|}, \qquad V_z = V\frac{r_z}{|r|},$$

Outputvar[i][3] X component of the velocity, magnitude [m/s].

$$= \sqrt{\mathrm{Re}(V_X)^2 + \mathrm{Im}(V_X)^2}$$

Outputvar[i][5] Y component of the velocity, magnitude [m/s].

$$= \sqrt{\mathrm{Re}(V_Y)^2 + \mathrm{Im}(V_Y)^2}$$

Outputvar[i][7] Z component of the velocity, magnitude [m/s].

$$= \sqrt{\mathrm{Re}(V_Z)^2 + \mathrm{Im}(V_Z)^2}$$

Outputvar[i][9] Magnitude of the velocity, magnitude [m/s].

$$= V$$

Outputvar[i][4] X component of velocity, phase [degrees].

$$= 180\left(\arctan\left[\frac{\mathrm{Im}(V_X)}{\mathrm{Re}(V_X)}\right]\right)/\pi$$

Outputvar[i][6] Y component of velocity, phase [degrees].

$$= 180\left(\arctan\left[\frac{\mathrm{Im}(V_Y)}{\mathrm{Re}(V_Y)}\right]\right)/\pi$$

Outputvar[i][8] Z component of velocity, phase [degrees].

$$= 180\left(\arctan\left[\frac{\mathrm{Im}(V_Z)}{\mathrm{Re}(V_Z)}\right]\right)/\pi$$

The acoustic intensity can be determined for each of these coordinate axes through the power dissipation analogy in electrical components. The intensity may be written in terms of the pressure and velocity using $I = p\breve{V} = \frac{1}{2}\mathrm{Re}\{P_{DRM}\breve{V}\}$ where the overbar denotes the complex conjugate.

Outputvar[i][10] X component of the intensity.

$$= \frac{1}{2}\mathrm{Re}\{P_{DRM}\breve{V}_X\}$$

Outputvar[i][11] Y component of the intensity.

$$= \frac{1}{2}\mathrm{Re}\{P_{DRM}\breve{V}_Y\}$$

Outputvar[i][12] Z component of the intensity.

$$= \frac{1}{2}\mathrm{Re}\left\{P_{DRM}\breve{V}_Z\right\}$$

THE SOLUTION METHOD FOR A MATRIX PROBLEM

Two numerical methods are available for the solution of an matrix problem, however, the computer code has been written so that a different solution method may be substituted with no significant alteration required. The two methods are Gaussian Jordan and LU decomposition, both validated through the use of complex value matrices.

The method is based on the classical equation $\mathbf{F} = \mathbf{A}\boldsymbol{\sigma}$ where the vector of boundary conditions and sound source contributions are given by $\mathbf{F}$ (known), the amplitudes of the fundamental solutions $\boldsymbol{\sigma}$ (unknown) and the square matrix of boundary integrals $\mathbf{A}$ (known).

In the code, these variables are copied into duplicated variables for processing. The solution function is then called, which replaces the square matrix $\mathbf{A}$ with $\mathbf{A}^{-1}$ and the vector $\mathbf{F}$ with $\boldsymbol{\sigma}$.

Details of solution methods are available in Flowers, "An Introduction to Numerical Methods in C++", Clarendon Press, Oxford, 1995.

The Gauss-Jordan elimination is not the most efficient solver, however, is well known and many varieties of the code exist. It differs from a standard Gauss solver as back substitution to reduce a matrix to a triangular form is avoided. Instead, additional operations reduce the matrix to diagonal formation. It is assumed that the matrix $\mathbf{A}$ is non-singular with a well defined inverse.

VII. INPUT FILE DEFINITION

The input files can be generated automatically using the software Sonitus Pre-BE, which formats the geometry and composes an appropriate structured mesh. As an alternative, the Matlab code for defined geometry can be utilised for the mesh generation.

Figure 12 Generation of a surface mesh using computer aided design tools

In order that users may determine their own software for input file generation, sample input files are provided showing the order of the file structure. The input data file (.dat) has the parameters described in Table 6.

DESCRIPTION OF FILE INFORMATION.
File heading, including creation time and date.
A list of fixed frequencies the solver will excite the domain with.
Information lines [4].
Solution type (indirect or direct).
Tolerances for the Sonitus Pre-BE geometry creator.
Reference values for the pressure, intensity and work.
Nodal numbers and positions (x,y,z).
Number and coordinates of the data recovery points (x,y,z).
Details of how the nodes join together to create elements (triangular or quadrilateral).
Details of how the nodes of the data recovery mesh join together to create elements.
Material description: what is the domain made of, density, speed of sound, real and imaginary terms are applicable here in the case of damping being studied.
Boundary conditions, velocities or velocity slopes applied to each structural element.
Monopole source location and strength.
Display information on the elements.
Sonitus Pre-BE display standards.
Input file end line.

Table 6 Input file description

VIII. OUTPUT FILE DEFINITION

The results file is generated automatically using the software Sonitus Post-BE, which loads the results file (*.rslt), and displays the data on the screen. A Matlab code is also available for this purpose, in order to allow integration with other CAE programs.

In order that users may determine their own results analysis software, sample results files are provided showing the order of the file structure. The output data file (.rslt) has the information provided in Table 7.

DESCRIPTION OF FILE INFORMATION.
File heading including generation date and time.
Number of frequencies which have been analysed.
A list of the frequencies together with the solution type (direct or indirect).
For each frequency, a list of the following parameters are provided for every data recovery point.
Sound Pressure Level (dB).
Acoustic Pressure, Magnitude.
Acoustic Pressure, Phase Angle.
X-Component of Velocity, Magnitude.
X-Component of Velocity, Phase Angle.
Y-Component of Velocity, Magnitude.
Y-Component of Velocity, Phase Angle.
Z-Component of Velocity, Magnitude.
Z-Component of Velocity, Phase Angle.
Magnitude of Velocity.
X-Component of Intensity.
Y-Component of Intensity.
Z-Component of Intensity.
Output file end line.

Table 7 Output file description

IX. PROGRAM OVERVIEW

SonitusBE.cpp

Syntax is SonitusBE INPUTFILE OUTPUTFILE

The Sonitus BE program code is comprised of a main file (SonitusBE.cpp) and a series of header files supplying generic subroutines. The pseudo-code program listing is as follows:

Display welcome message to the screen.
Ensure correct number of arguments are passed at the command line.
Argument one is the input filename, two is the output filename, three is an unused protocol identifier.
Obtain input and output filenames from the command line arguments.
Call the routine to read in all input data, positions, boundary conditions and source locations.
Call the routine to check the status of the input mesh for consistency.
Call the routine to check that the required number of boundary conditions have been supplied.
Generate an empty list of output result variables.
Generate the output file (.rslt format).
Call the routine holding the numerical solver (direct or indirect solver).
Repeat for each excitation frequency.
Print output messages to the screen.
Release memory and close program safely.

The primary namespaces for the boundary element solver are SonitusBem and BemRoutine

X. SUBROUTINE FUNCTIONS

The subroutine header files contain a series of both generic and specific functions which manipulate the data in different ways. In this section, the individual routines are discussed and function calls detailed.

ErrorHandle.h (COMPLETE)

This file contains a routine which manages the error messages sent to it, and uses a built in function to print them to the screen.

```
BemRoutine::GeneralException(const string &mess){message=mess;}
```

Interp.h (COMPLETE)

This file contains the routines which interpolate data from numerical recipes type functions. It exists to contain all of the routines which allow interpolation on triangular and quadrilateral elements.

The primary namespace for the routines are Recipes, as the functions are derived both from the Numerical Recipes formulations, and formulae provided in Abramowitz and Stegun eds. (1972), *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, New York: Dover Publications, ISBN 978-0-486-61272-0.

The following routines are provided based on an interpolation of double precision numbers.

- Interpolation routine for a triangular element with nodes N_1 to N_3 and non-dimensionalised surface coordinates ζ_1 and ζ_2, which vary from 0 to 1.

```
double interpTRI( const double &N1,
                  const double &N2,
                  const double &N3,
                  const double &zeta1,
                  const double &zeta2);
```

Define a set of weighting functions, ψ_1, ψ_2, ψ_3, equated to $\psi_1 = \zeta_1$, $\psi_2 = \zeta_2$, $\psi_3 = 1 - \zeta_1 - \zeta_2$.
Interpolated return value = $\psi_1 N_1 + \psi_2 N_2 + \psi_3 N_3$.

- Interpolation routine for a triangular element with nodes N_1 to N_3 and non-dimensionalised surface coordinates ζ_1 and ζ_2 which vary from 0 to 1. This provides the derivative $\partial u / \partial \zeta_1$ at the values of ζ_1 and ζ_2.

```
double DZETA1interpTRI(
                  const double &N1,
                  const double &N2,
                  const double &N3,
                  const double &zeta1,
                  const double &zeta2);
```

Return $\partial u / \partial \zeta_1$ = N1-N3.

- Interpolation routine for a triangular element with nodes N_1 to N_3 and non-dimensionalised surface coordinates ζ_1 and ζ_2 which vary from 0 to 1. This provides the derivative $\partial u/\partial \zeta_2$ at the values of ζ_1 and ζ_2.

```
double DZETA2interpTRI(
                    const double &N1,
                    const double &N2,
                    const double &N3,
                    const double &zeta1,
                    const double &zeta2);
```

Return $\partial u/\partial \zeta_2$ = N2-N3.

- Interpolation routine for a quadrilateral element with nodes N_1 to N_4 and non-dimensionalised surface coordinates ζ_1 and ζ_2 which vary from -1 to 1.

```
double interpQUAD(const double &N1,
                    const double &N2,
                    const double &N3,
                    const double &N4,
                    const double &zeta1,
                    const double &zeta2);
```

Define a set of weighting functions, ψ_1, ψ_2, ψ_3, ψ_4 equated to $\psi_1 = \frac{1}{4}[(1-\zeta_1)(1-\zeta_2)]$, $\psi_2 = \frac{1}{4}[(1+\zeta_1)(1-\zeta_2)]$, $\psi_3 = \frac{1}{4}[(1+\zeta_1)(1+\zeta_2)]$, $\psi_4 = \frac{1}{4}[(1-\zeta_1)(1+\zeta_2)]$.

Interpolated return value = $\psi_1 N_1 + \psi_2 N_2 + \psi_3 N_3 + \psi_4 N_4$.

- Interpolation routine for a quadrilateral element with nodes N_1 to N_4 and non-dimensionalised surface coordinates ζ_1 and ζ_2 which vary from -1 to 1. This provides the derivative $\partial u/\partial \zeta_1$ at the values of ζ_1 and ζ_2.

```
double DZETA1interpQUAD(
                    const double &N1,
                    const double &N2,
                    const double &N3,
                    const double &N4,
                    const double &zeta1,
                    const double &zeta2);
```

Define a set of weighting functions, ψ_1, ψ_2, ψ_3, ψ_4 equated to $\psi_1 = \frac{1}{4}[-1+\zeta_2]$, $\psi_2 = \frac{1}{4}[1-\zeta_2]$, $\psi_3 = \frac{1}{4}[1+\zeta_2]$, $\psi_4 = \frac{1}{4}[-1-\zeta_2]$.

Interpolated return value = $\psi_1 N_1 + \psi_2 N_2 + \psi_3 N_3 + \psi_4 N_4$.

- Interpolation routine for a quadrilateral element with nodes N_1 to N_4 and non-dimensionalised surface coordinates ζ_1 and ζ_2 which vary from -1 to 1. This provides the derivative $\partial u/\partial \zeta_2$ at the values of ζ_1 and ζ_2.

```
double DZETA2interpQUAD(
                    const double &N1,
```

```
                    const double &N2,
                    const double &N3,
                    const double &N4,
                    const double &zeta1,
                    const double &zeta2);
```

Define a set of weighting functions, ψ_1, ψ_2, ψ_3, ψ_4 equated to $\psi_1 = \frac{1}{4}[-1+\zeta_1]$, $\psi_2 = \frac{1}{4}[-1-\zeta_1]$, $\psi_3 = \frac{1}{4}[1+\zeta_1]$, $\psi_4 = \frac{1}{4}[1-\zeta_1]$.

Interpolated return value = $\psi_1 N_1 + \psi_2 N_2 + \psi_3 N_3 + \psi_4 N_4$.

A similar set of routines are also provided for the case where the interpolated values are complex and of double precision.

```
ComplexMath_std::complex <double> CinterpTRI(
                    const ComplexMath_std::complex <double> &N1,
                    const ComplexMath_std::complex <double> &N2,
                    const ComplexMath_std::complex <double> &N3,
                    const double &zeta1,
                    const double &zeta2);

ComplexMath_std::complex <double> CinterpQUAD(
                const ComplexMath_std::complex < double > &N1,
                const ComplexMath_std::complex < double > &N2,
                const ComplexMath_std::complex < double > &N3,
                const ComplexMath_std::complex < double > &N4,
                const double &zeta1,
                const double &zeta2);
```

Messages.h (COMPLETE)

The primary namespace for this set of subroutine functions is SontitusBem.

This file contains the routines which print messages of welcome to the screen and to an output file.

There are three functions provided which print information to the control console screen, showing the requirements of the program, the welcome message with copyright information and a final screen which lets the user know that the program has finished running for all frequencies.

```
    void requirements(void);
    void welcome(void);
    void goodbye(void);
```

SolutionChoice.h (COMPLETE)

The primary namespace for this routine is BemRoutine.

This file returns the method used in the boundary element calculation. Only the indirect method is implemented in this code.

The only function call is to determine what solver is used, either direct formulation or indirect. At this time, the only function only returns the indirect solver choice, regardless of the input

conditions. If returned integer is 1, the direct solver would be chosen, if 2 the solver is indirect formulation.

```
int SolutionMethod(void);
```

Verification.h (COMPLETE)

The primary namespace for this routine is SonitusBemFormat.

This file checks the meshes for consistency.

The function checkmesh provides basic error checking for the input structural mesh, in order to ensure that the correct number of nodes, elements, boundary conditions are provided, as without these the solver would be unable to reliably complete the calculation and would perform in an unpredictable fashion.

```
void checkmesh(const int & NumberofNodes,
     const vector <int> & NodeNum,
     const vector <MeshTypes::Node <double> > &NodeCoord,
     const vector <MeshTypes::Element <int> > &ElemStruct,
     const vector <int> &ElemNum,
     const int &NumofElems)
```

This routine checks for repeated node numbers in the same element. It also compares sequential node numbers in different elements to ascertain whether the nodes are all pointing in consistent directions. Essentially, the element direction is determined by the order in which the nodes are arranges, either clockwise or counter-clockwise. If two different elements have two of the same nodes in the same direction, then the two elements are facing in opposite directions.

This routine contains a series of if then else statements which test the above condition, whilst taking into account the possibility of triangular or quadrilateral element types.

```
void checkBCnumbers(const int &NumofElems,
     const int &BCnum)
```

This routine compares the number of elements and the number of boundary conditions available. If these are the same, then we have a fully constrained problem and can send the problem to the boundary element solver for completion. If not, these we have a badly posed input problem and must correct the input files before proceeding.

Structures.h (COMPLETE)

The primary namespace for these subroutines is MeshTypes.

Routine allows vectors to be manipulated.

This routine contains the definitions of different classes used in the main program.

Definition of a node, with three double elements representing (x, y, z).

Definition of an element, defined by five fields (integers). The first field is a flag which says what element type is described.

If the flag is 1, this means a triangular element, and the following three fields are integers containing the node numbers defining that triangle.

If the flag is 2, this means a quadrilateral element, and the following four fields are integers containing the node numbers defining the corners of the quadrilateral.

Definition of a boundary condition number class. This has two entries, the first is the boundary element number (this should coincide with the element number) and the second is the type (1 is velocity and 2 is impedance).

FileReaderSonitus.h (COMPLETE)

The primary namespace for this routine is SonitusBemFormat.

This file contains the routines which read in the data from an input file.

The routine call is as follows. Input the filename as a string and it will find the entries for the input data file.

```
void FilereaderSonitus::getinputdata(const string filename)
```

The routine by checking whether the input file can actually be opened or whether the routine needs to terminate an report a failure to complete. If no errors are reported, it means we can then generate and initialise variables.

The first seven lines are assumed to be organised header information describing the kind of input geometry, so that the user can keep track of what the file is representing.
Note that the maximum length of any input data line is 200 characters. Any lines exceeding this will cause a termination error.

The next lines should start with the word ACOUSTIC, then listing the acoustic frequency to investigate using the boundary element solver. Then read in and interpret control flags and data control cards.

Next the reference pressure (PREF), reference intensity (IREF) and reference work (WREF) are read in. The next four lines are then ignored.

The nodal coordinates are then read in, with each line starting with the word NODE. The node number is found, then the three coordinates interpreted. The next four lines are ignored.
The number and location of the data recovery nodes are then read in, with each line starting with DRNODE. The next four lines are ignored.

Once the nodes are known and organised, the elements can be defined. A triangular element is defined through three nodes, while a quadrilateral requires four nodes.
The data entry for a triangular element starts with the word TRIA3. If the triangular elements are not found, check for the word QUAD4, and read in the nodes which create these. The next four lines are ignored.

Although the data recovery nodes do not need to be generated as a mesh, it is found that this is more useful for processing results over an area. Given this is a possibility, the program now looks for the elements which make up the data recovery surface. Again, for each element, the control word TRIA3 or QUAD4 defines the type of element. The next two lines are ignored. At this point, the geometric information has been completely read in. The next information which is required is the type of material that the acoustic waves are propagating through, whether it is air or liquid. The primary difference is the density, with the control line starting with the word MATERIAL. The

density has real and imaginary components, as does the subsequent speed of sound. The following six lines are then ignored.

The boundary condition information is then provided, such as whether for each element the boundary condition is a velocity (control word EVELO) or an impedance (EIMPE) and what value it takes.

Finally, in order to obtain a non-trivial solution, where no sound field is present, a source must be introduced into the calculation. At present, only one sound source can be introduced, although this may be modified should the requirement exist. The control word for this is SOURCE, followed by the source strength, including real and imaginary components, and the source location.

The routine ends by closing the input file and returning to the main program with the input data.

Resultoutput.h (COMPLETE)

The primary namespace is BemRoutine.

Outputs the file results including sound pressure over the data recovery mesh.

Once the boundary element solver has completed all of the calculations, it needs to output the results to a file which can be post-processed at a later date without the worry of losing information held in memory. Rather than using a proprietary format for the generation and storage of results, the output files are held in a plain text ASCII format.
There are three routines in this header file, these are:

```
void ResultWritingInit(const string &filename,
                const int &Nfreqs,
                const vector <double> &Freqs,
                const int &NumDRNodes,
                const vector <int> &DRNodeNums);

void ResultWritingFreq(const string &filename,
                const int &ifreq,
                const int &NumDRNodes,
                const vector <int> &DRNodeNums);

void ResultWritingFinish(const string &filename);
```

The first function opens a results file for outputting the results into, including checking whether the file is valid and successfully opened. At this point the "offline" information is written, including the date and time of the run, the number and value of the frequencies, whether the solver is an indirect or direct boundary element type. The file is then closed.

The second function opens the file again for each frequency (when the solver has generated a set of results at a specific frequency, this function is called). Then for each data recovery node, the sound pressure level in dB, the acoustic pressure (magnitude and phase angle), the (x, y, z) magnitude and phase of the velocities are written, the magnitude of the velocity, the (x, y, z) values of the intensity are output.

Finally, the third function opens the results file, places a termination character and closes the file again.

The default spacing for each entry is 12 characters using a 5 digit precision, should alterations be required, this is a simple modification to enact.

complex.h (COMPLETE)

The default namespace for this function is ComplexMath_std.

Provides complex number handling for a range of functions.

This provides handling for complex numbers, including returning real and imaginary components, performing addition, subtraction, multiplication, division and power calculations.

IndirectSolver.h (COMPLETE)

The primary namespace for this routine is BemRoutine.

This is the file which solves the boundary element matrix. It assembles all terms and requests the solution to the matrix problem.

This is the main routine which solves the boundary element acoustic problem. Given a surface region Gamma, with boundary conditions defined as velocity terms for each element [u1 u2 u3...]^T, we require the amplitude of source terms sigma on each element which cause the boundary conditions to be satisfied.
So ui = Gij sigma_j.

So the boundary condition at node i has to be satisfied with a distribution of sources at elements j. The weightings are given by fundamental solutions.

This header file contains a number of subroutines, which will be described in turn. The most significant is the indirect solver routine, as shown below.

```
void IndirectSolution(const double &Cspeed,
                      const double &rhodensity,
                      const int &Nfreqs,
                      const int & ifreq,
                      const double &Freq,
                      const double &Pref,
                      const double &Iref,
                      const int &NumNodes,
                      const vector <int> &NodeNum,
                      const vector <MeshTypes::Node <double> >
&NodeCoord,
                      const int &NumDRNodes,
                      const vector <int> &DRNodeNums,
                      const vector <MeshTypes::Node <double> >
&DRNodeCoord,
                      const int &NumofElems,
                      const vector <int> &ElemNum,
                      const vector <MeshTypes::Element <int> >
&ElemStruct,
```

```
                    const vector <MeshTypes::BCdetails <int>
                > &BCnum,
                    const vector < ComplexMath_std::complex
                    <double> > &BCval,
                    const ComplexMath_std::complex <double>
                    &ST,
                    const vector <MeshTypes::Node <double> >
                    &SL,
                    vector <double> &OutputdB,
                    vector <vector <double> > &OutputVar);
```

The matrices are initialised, for F = A Sigma, where F are the array of boundary conditions, Sigma is the vectors of unknown potentials and A the matrix of boundary integrals. For every element, we perform a boundary integral for every other element. As the elements are all at different angles to each other, we perform a Jacobian transformation based on non-dimensionalised coordinates on the surface of one element to obtain the integral.

For the non-trivial solution, we must also add the boundary integral contribution of a discrete point source, which is calculated using the IntBHelm function.

We then solve the system of equations and based on this solution, generate a series of physical parameters at every single data recovery point.

In order to calculate the integral term for each element, we call the routine IntSourceHelm. This can be described as finding the contribution to the node i from a source placed on element j. Therefore we are observing the contribution of element j. This is determined by examining the amplitudes of the fundamental solutions at each of the element j's node locations.

```
ComplexMath_std::complex <double> IntSourceHelm(
          const MeshTypes::Node <double> &iNode,
          const MeshTypes::Element <int> &ObsElemStruct,
          const MeshTypes::Node <double> &ObsNode1,
          const MeshTypes::Node <double> &ObsNode2,
          const MeshTypes::Node <double> &ObsNode3,
          const MeshTypes::Node <double> &ObsNode4,
          const double &Freq, const double &C,const double
     &rho);
```

The code first calculates the location of the node i. It then immediately branches into a loop depending on whether the element j is triangular or quadrilateral. This then follows the same concept described in section VI.
The value of u* for each node making up element j is obtained by examination of the distance between this element's node and the original i node.

For each node i, the contribution to the boundary condition value must be included. This routine calculates the required contribution of the pressure at node i from the source.

```
ComplexMath_std::complex <double> IntBHelm(
     const MeshTypes::Node <double> &iNode,
     const ComplexMath_std::complex <double> &Bsourcestr,
     const MeshTypes::Node <double>  &Bsourceloc,
```

```
          const double &Freq, const double &speed,const double
     &rho);
```

The location of the source is provided by the variables `(Bsourceloc.x, Bsourceloc.y, Bsourceloc.z)`. The distance between the source and the node at location i is then,

```
r=(  (Bsourceloc.x-iNode.x)^2
+    (Bsourceloc.y-iNode.y)^2
+    (Bsourceloc.z-iNode.z)^2)^0.5;
r = abs(r);
```

$B = Bsourcestr\ \omega\rho * \text{HelmFunSol(r,speed,Freq)}$

The fundamental solution is provided through the HelmFunSol function, as follows.

```
     ComplexMath_std::complex <double> HelmFunSol( const double
&R,
          const double &speed,
          const double &f);

     void IntV(
          const MeshTypes::Node <double> &iNode,
          const ComplexMath_std::complex <double> &Bsourcestr,
          const MeshTypes::Node <double>  &Bsourceloc,
          const double &Freq, const double &speed,const double
     &rho,
          ComplexMath_std::complex <double > &Velx,
          ComplexMath_std::complex <double > &Vely,
          ComplexMath_std::complex <double > &Velz,
          ComplexMath_std::complex < double > &V);
```

This provides

$$u^* = \frac{1}{4\pi R}\exp\left(-i2\pi FR/C\right)$$

Makefile (COMPLETE)

This file contains the make instructions for the overall program executable. Currently includes options for operating system specific choices, i.e. Windows or Linux directives.

DJO_LU.h (INCOMPLETE)

This routine is designed based on the Numerical Recipes subroutines to solve the system of equations A x = (L U) x = L (U x) = B.

L y = B
U x = y
Ax = b

We assume we know the square matrix A[0..n-1][0..n-1] and also the vector b[0..n-1]. We return the inverse matrix A^-1 and the solution vectors x.

Require assessment.

This routine performs an LU decomposition on a matrix. This is a slightly more numerically stable method of doing an analysis rather than inverting the main matrix. Given a typically square matrix a[1..n][1..n] this routine replaces it by the LU decomposition of a row-wise permutation of itself. The matrix A and and size n are input, a is output. indx[1..n] is an output vector that records the row permutation effected by the partial pivoting; d is output as +- 1 depending on whether the number of row interchanges was even or odd, respectively. This routine is used in combination with lubksb to solve linear equations or invert a matrix.

```
void DJOLudcmp( vector <vector <double> > &A,
     const int n, vector <int> &indx,double d)

void DJOLubksb( vector < vector <double> > &A,
     const int n, vector <int> &indx, vector <double> &B)
```

The versions of these for complex variables and matrices are provides also with the same input format.

```
     void DJOLudcmpc( vector <vector <ComplexMath_std::complex <
double> > > &A,
          const int n, vector <int> &indx, double d)

     void DJOLubksbc( vector < vector <ComplexMath_std::complex <
double> > > &A,
          const int n, vector <int> &indx,
          vector <ComplexMath_std::complex < double > > &B)
```

Gaussjc.h (COMPLETE)

This is a simple Gauss Jordan solver based on Numerical Recipes method for solving systems of equations of the type Ax = b.

We assume we know the square matrix A[0..n-1][0..n-1] and the vector b[0..n-1]. We return the inverse matrix A^-1 and the solution vectors x.

Require assessment.

```
     void Gaussjc( vector <vector <ComplexMath_std::complex <
double > > > &A,
          const int n,
          vector <ComplexMath_std::complex < double> > &B)
```

XI. SAMPLE FILES

Three sets of results are provided for illustration of the capability of the Sonitus BE software. These are all based on a solid tyre geometrical shape, resting on an infinite plane. The data recovery mesh is also based on the automotive case. The first case is where a source is placed in the far-field and the surface pressure is calculated when there is no solid geometry anywhere in the domain. This relatively trivial solution, replicates the standard Green's function solution for a monopole source propagating outwards, with a mirror image source correcting the result for the ground plane.

The second result is the case where the solid tyre geometry is input. At this point a sound amplification is possible due to the diverging surfaces near to the contact patch where amplifications of up to 20dB are possible for certain frequencies.

The final case is the sound pressure inside a tyre surface with a sound source and a damping layer (modelled as a surface with a prescribed complex impedance). This not only demonstrates the use of the solver for interior against exterior domains, but also demonstrates the ability to mix boundary conditions, with different surfaces containing different damping materials.

Solid geometry

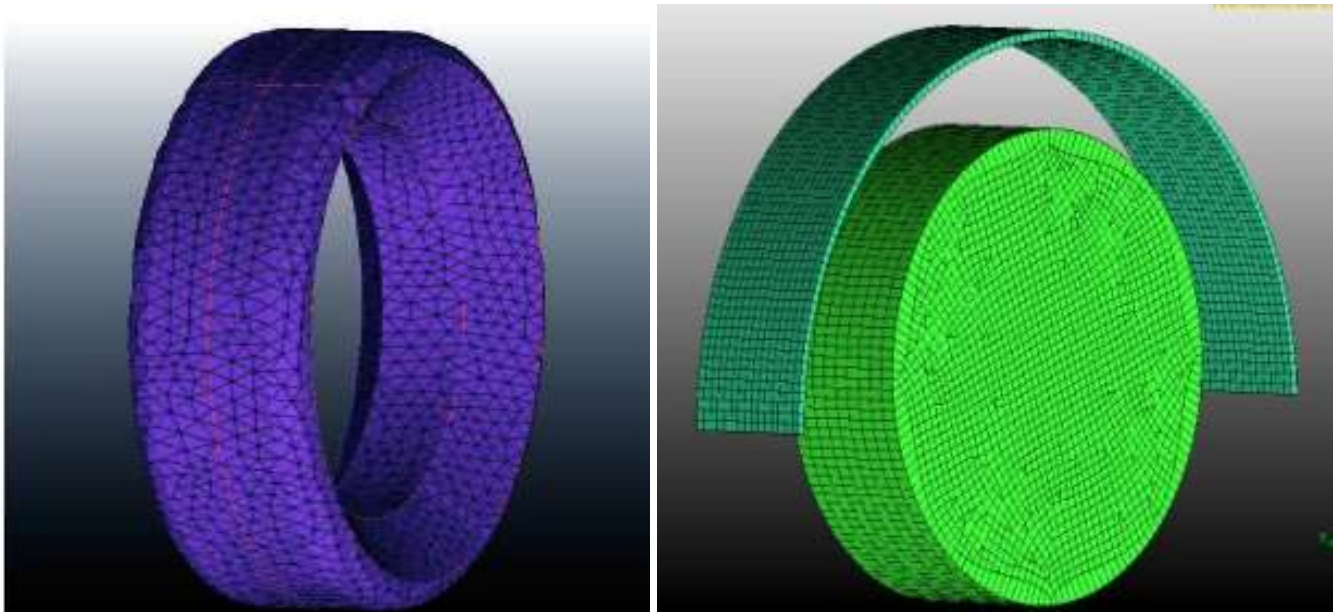

Figure 13 Examples of solid geometry defined by an unstructured surface mesh composed of triangular elements and one with a mixture of quadrilateral structured elements (on the wheel arch) and a mixture of triangular and quadrilateral elements on the wheel surface.

Infinite plane surface

The solver has the option (through the input data file) to allow the incorporation of an infinite plane to provide a ground plane for any solid geometry. A finite area section of this is shown in Figure 14 where the normal impedance relation between pressure and velocity can be prescribed as a complex value.
The use of a complex impedance allows the efficient inclusion of a ground surface which has variable levels of acoustic damping.

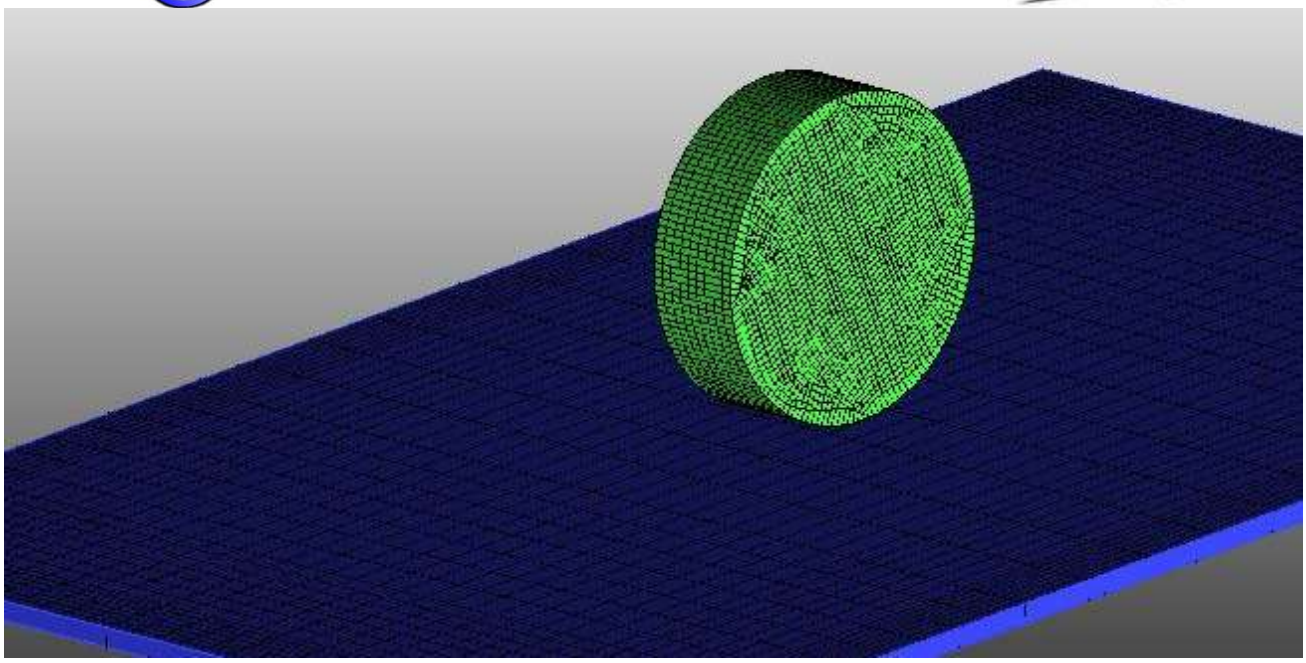

Figure 14 Ground plane (similar to an image source)

Data recovery mesh (exterior)

The data recovery array is a series of nodes in the xyz domain, see Figure 15.

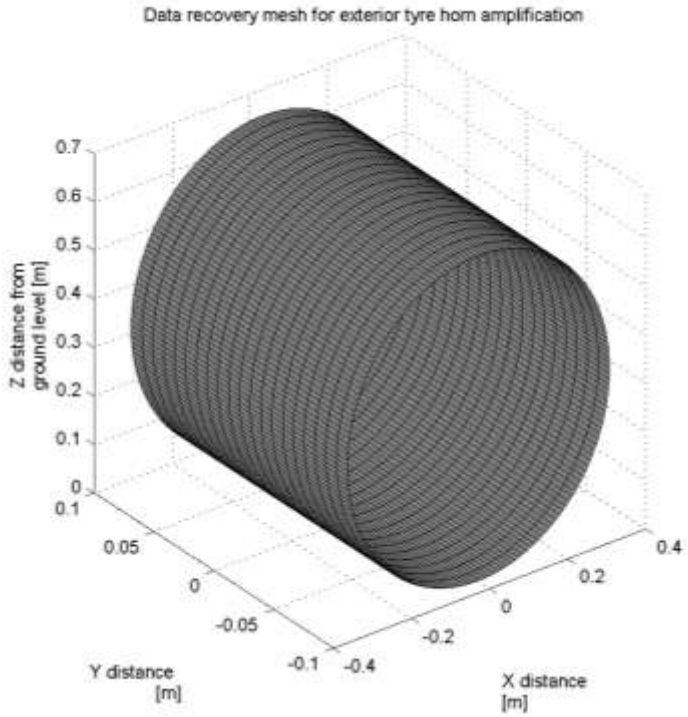

Figure 15 Data recovery mesh for automobile tyre noise

Data recovery mesh (interior)

Free-field solution

The free-field solution is a relatively trivial case, where the solid geometry in the exterior domain is extremely small, (a cube surface comprised of fifty nodes is used as a basis for the solution), with quadrilateral regular mesh spacing for integration of surface integrals. The tyre data recovery mesh shown in Figure 15 is used to generate the sound pressure for a range of excitation frequencies between 0.1 and 2kHz for a source located at (X,Y,Z)=(1.0,0.0,0.1)m..

The strength of the source is unity, providing a relatively large sound pressure at the data recovery positions, which are fairly close to the source position. The sound pressure at an excitation frequency of 100, 400 and 1300 Hz are provided in Figure 16.

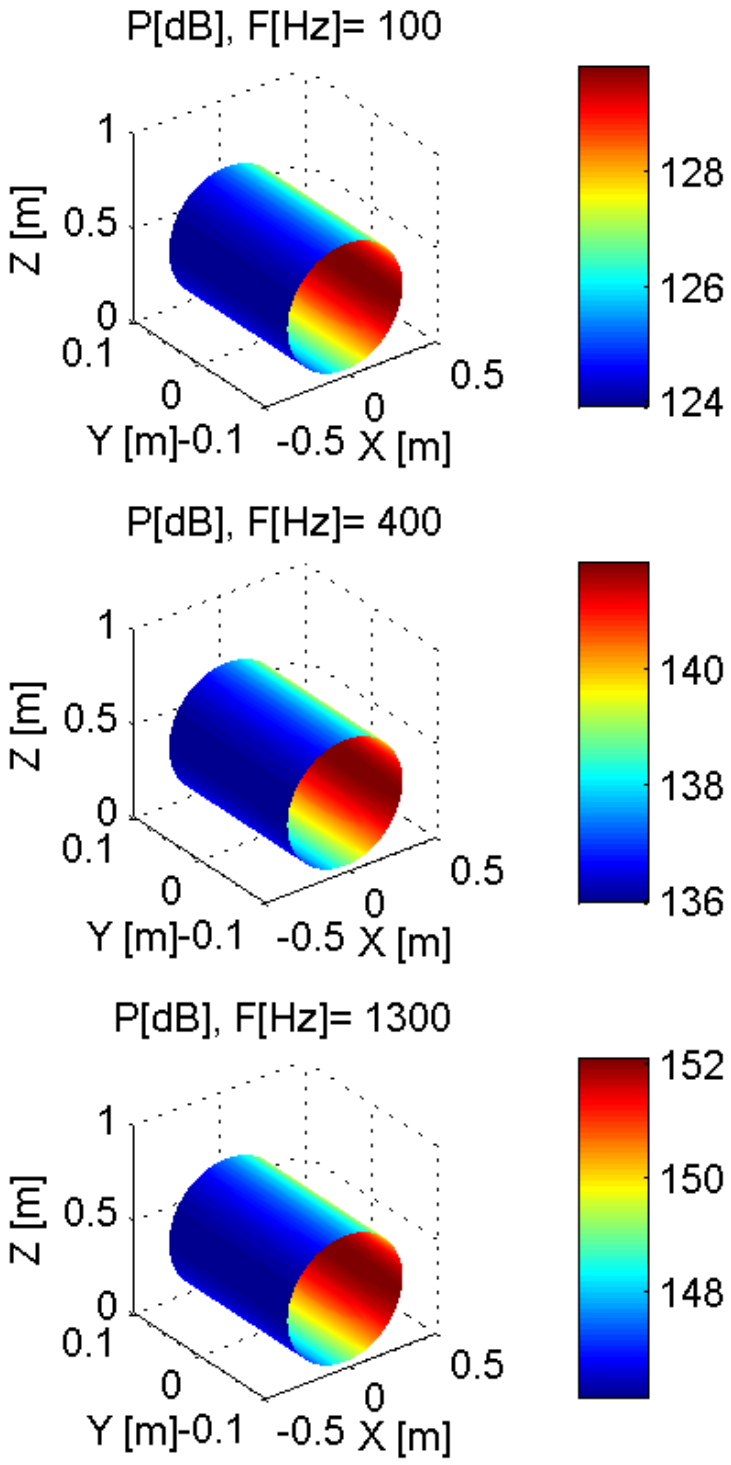

Figure 16 Sound pressure for a source in the free-field (note the scale of the amplitude is different at each frequency).

Sound pressure in a domain with a solid geometry

The solution for a sound source in a free field with no solid geometry leads to a spherically propagating pressure distribution. As soon as a solid geometry is introduced into the environment, it becomes possible for the sound field to scatter, leading to regions where for some frequencies, an amplification effect is possible.

In this example, the amplification due to the geometry of a tyre is investigated, as the tyre surface diverges from the road in a pattern similar to a gramophone horn. When the tyre vibrates close to the road surface, this leads to sound close to the contact patch at the throat of the horn. Any sound source is therefore amplified, see Figure 17.

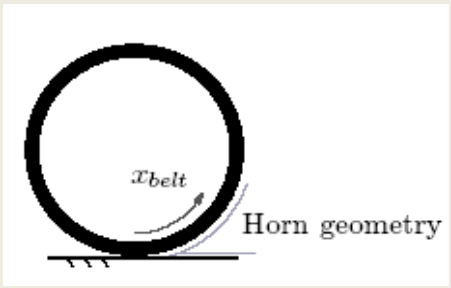

Figure 17 Diverging surfaces leading to a horn geometry where sound sources may be amplified

In this case, rather than placing the sound source in the horn area, it is placed in the far-field and the tyre surface becomes the observer point. This reciprocal technique is used to simplify the number of calculations required.

The sound pressure at an excitation frequency of 100, 400 and 1300Hz are shown in

Figure 18 Sound pressure for a source with a solid geometry in the domain (note the scale of the amplitude is different at each frequency).

Sound pressure inside an enclosure with damping material included on one surface

XII. NOTES

Exterior and interior domains

The solver code can calculate the sound pressure in an enclosed domain or an infinite domain (where we assume Sommerfeld's radiation condition holds). However, it is essential that in the creation of the solid geometry mesh and data recovery meshes, that the direction of element normals are consistent (either all pointing into the domain or out of the domain). The code provides a check to ensure that the elements are all consistent with each other and will generate an error message should this not be the case.

Element types
There are two main types of geometrical element which generate a surface area. These are triangular and quadrilateral. The solver can integrate both of these and the choice is largely dictated by the mesh generation software, the mesh density and shape of the solid geometry.

Sound sources
A number of monopoles can be prescribed at different locations, with different strengths, to generate an excitation sound field. The amplitude of each source can be complex, allowing a different phase relation to be produced between different sources.

Infinite ground plane
In order to simulate a ground surface, an infinite impedance ground plane can be introduced into the calculation, similar to the effect of adding image sources into the model.

XIII. INPUT FILE TYPES - DAT

FreeField.dat
21 Frequencies
50 Nodes
4160 Data recovery nodes
Quad elements for structure
Quad data recovery elements
Density 1.12kgm-3
Speed of sound 343m/s
Zero velocity boundary conditions
SOURCE 1 1.00000E+000 0.00000E+000 1.00000E+000 0.00000E+000 1.00000E-001

XIV. DEBUG REQUIREMENTS

Debug problems:

i) Windows version of the LU decomposition requires checking with new maths header files.
ii) Sample files need validating for inclusion in final release. Freefield.dat needs two more elements to ensure an enclosed domain.
iii) Pre-and Post processing scripts.
iv) The velocity term in the boundary condition specifies the magnitude of the velocity on the element. It could be an improvement to only specify the normal velocity on the element.
v) The contribution of the mesh to the velocity in the data recovery section is not coded at this time. Therefore, the only contribution is from the monopole source.

XV. SUMMARY

This manual has provided the details of a numerical solver for a boundary element solver code designed specifically for acoustics applications. Although the method is similar to many other engineering applications, for example, heat transfer, the variables are all related to acoustic fields.

The Sonitus BE acoustic solver provides a comprehensive tool for the investigation of acoustic problems within a wide range of industrial sectors. This manual provides a selection of the numerical techniques used to produce this program. The manual sets out the physical processes which occur in the code, and provides guidance on the physical parameters being calculated.

Sample files have been provided so that a user may write their own input geometry and mesh files for processing. For further information, see the contact details at the beginning of this document.

XVI. REFERENCES

C. A. Brebbia, "*The Boundary Element Method for Engineers*". Pentech Press, 1980.

C. A. Brebbia and S. Walker, "*Boundary Element Techniques in Engineering*", Newnes-Butterworths, 1980.

Abramowitz and Stegun eds. (1972), *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, New York: Dover Publications, ISBN 978-0-486-61272-0.

F. Fahy and P. Gardonio, "*Sound and structural vibration*". Academic Press, 1985.

K. F. Graff, "*Wave Motion in Elastic Solids*". Clarendon Press, Oxford, 1975.

D. G. Crighton, A. P. Dowling, J. E. Ffowcs Williams, M. Heckl, and F. G. Leppington, "*Modern methods in Analytical Acoustics*". Springer-Verlag, 1992.

M. Heckl, L. Cremer and E. E. Ungar, "*Structure Borne Sound*", 2nd Edition. Springer-Verlag, Berlin, 1988.

D. E. Bourne and P. C. Kendall, "*Vector Analysis and Cartesian Tensors"*, 3rd Edition. Chapman and Hall, 1992

A. P. Dowling and J. E. Ffowcs Williams, "*Sound and Sources of Sound*". Ellis Horwood, 1989.

W. H. Press, W. T. Vetterling and B. P. Flannery, "*Numerical recipes in Fortran: the art of scientific computing*". Cambridge University Press, 1993.

M. R. Spiegel, "*Mathematical Handbook of Formulas and Tables*". Schaum, 1998.

E. Kreyszig, "*Advanced Engineering Mathematics*", 8th Edition. Wiley, 1998.

Flowers, "*An Introduction to Numerical Methods in C++*", Clarendon Press, Oxford, 1995.

A

Abramowitz and Stegun, 37, 54
acoustic, 11, 13, 16, 17, 18, 20, 22, 41, 42, 43, 47
Acoustic Pressure, 36
amplitude, 17, 18, 19, 20, 21, 43, 50, 51, 52
ASCII, 42

B

BEM, 11
boundary, 11, 13, 16, 17, 18, 19, 20, 21, 22, 26, 28, 36, 39, 40, 41, 42, 43, 44, 47, 52

C

CAD, 11, 13
complex, 13, 17, 39, 43, 44, 45, 46, 47, 52

D

Data recovery mesh, 18, 48
discretisation, 13, 22
domain, 11, 13, 16, 17, 18, 19, 20, 21, 22, 24, 35, 46, 48, 49, 51

E

Element, 1, 11, 16, 20, 40, 43, 44, 52, 54
error, 13, 21, 37, 40, 41, 51
Excitation frequency, 17

F

finite element, 11, 13, 19
Fundamental solution, 17

G

Gauss Jordan solver, 46
Green's function, 14, 17, 21, 46

H

Helmholtz, 17, 18, 21, 22

I

impedance, 13, 41, 42, 47, 52
Input, 33, 35, 41, 52
Interpolation, 37, 38

J

Jacobian transformation, 44

L

Linux, 11, 12
LU decomposition, 46

M

MATERIAL, 41
Matlab, 11, 33, 35
monopole, 17, 18, 19, 46

N

Node, 16, 40, 43, 44, 45
Numerical Recipes, 37, 45, 46

O

Output, 35, 36

P

phase, 18, 42, 52
potential, 14
precision, 37, 39, 43
pressure, 11, 17, 18, 19, 20, 33, 35, 42, 44, 46, 47, 49, 50, 51

Q

quadrilateral, 16, 35, 37, 38, 41, 47, 49, 52

R

reference intensity, 41
reference pressure, 41
reference work, 41
Requirements, 12, 14, 15, 33, 35, 36

S

Sample files, 46
SOURCE, 42, 52
Subroutine, 37
surface, 11, 13, 14, 16, 17, 18, 19, 20, 21, 22, 24, 34, 37, 38, 41, 43, 44, 46, 47, 49, 51, 52
Surface, 16

T

triangular, 16, 35, 37, 38, 40, 41, 47, 52

V

velocity, 11, 17, 19, 35, 41, 42, 43, 47, 52
Velocity, 36

W

weighting, 14, 17, 21, 37, 38, 39
Windows, 11